Universidad Politécnica de Madrid

Facultad de Informática

Proyecto fin de carrera

Diseño e implementación de una aplicación web para la gestión y control de visitas a centros educativos

Autor: David Fernández Moreno
Tutores: Víctor Robles Forcada
Oscar Cubo Medina

Madrid, Noviembre 2006

Diseño e implementación de una aplicación web para la gestión y control de visitas a centros educativos

David Fernández Moreno

Madrid, Noviembre 2006

Tutores:

Víctor Robles Forcada (vrobles@fi.upm.es)

Oscar Cubo Medina (ocubo@fi.upm.es)

Dirección:
Facultad de Informática – Universidad Politécnica de Madrid
Campus de Montegancedo s/n
Boadilla del Monte
28660, Madrid (España)

Teléfono:
(+34) 91 336 74 50

Correo electrónico:
dfernandez@laurel.fi.upm.es

Página web:
http://laurel.datsi.fi.upm.es/∼dfernandez/

La composición de este documento se ha realizado con LaTeX.
Diseño de David Fernndez Moreno.

Hay varias clases de curiosidad: una, interesada, que nos lleva a desear aprender lo que nos puede ser útil; otra, orgullosa, nacida del deseo de saber lo que otros ignoran...

François de la Rochefoucauld

A todos los que siempre han confiado en mi y en especial a mi familia

Sinópsis / Abstract

Sinópsis

Dentro del proceso de promoción institucional de la Universidad Politécnica de Madrid (UPM), se realizan visitas informativas a distintos centros educativos de la comunidad. El proyecto surge a partir de la necesidad de gestionar estas visitas.

Para la realización del mismo, se ha desarrollado una aplicación web utilizando diferentes tecnologías, como han sido *Hypertext Preprocessor* (PHP), AJAX y *Hyper Text Markup Language* (HTML).

El desarrollo de esta aplicación comprende una parte de diseño y otra de implemetanción. En la parte de diseño se ha utilizado una metodología orientada a objetos.

Agradecimientos

”..El bien hacer abre cien puertas, y el mal agradecer las cierra...”

Y por lo tanto, y desde aquí, quiero agradecer a todas las personas, que de una forma u otra, han estado conmigo durante todos estos años, unas veces mas cerca y otras menos.. Compañeros a los que no me canso de ver, otros a los que hace mucho tiempo que no veo y algunos otros a los que a lo mejor nunca volveré a ver, pero de los que guardo un maravilloso recuerdo. Gente que ha pasado muy deprisa por mi vida y personas con las que quiero pasar el resto de ella y a quienes voy a estar eternamente agradecido.

Muchas de estas personas tienen nombres propios y en las siguientes líneas quiero acordarme de ellas.

Por ello, y en primer lugar, quiero agradecer a mi tutor; Víctor Robles Forcada, por todo lo que aprendí durante el tiempo en el que cursé la asignatura de Sistemas Informáticos y por haber confiado en mi para llevar a cabo este proyecto.

Quiero acordarme también de todos los compañeros que he tenido durante la carrera; llámense Pablete, Josete, Gonzalo, Javi, Micky, Luis, Ana, Carlos, Juanpi, Ricky, y un largo etcétera que aunque no nombre son igual de importantes. Compañeros de laboratorio, con lo que por desgracia he pasado poco tiempo y dentro de ellos, uno en especial, Oscar Cubo Medina. Quiero agradecerte, ya no por tantísimas cosas que he aprendido contigo sino por todos los ratos que hemos pasado, de corazón, gracias por todo lo que has hecho por mi, porque aunque no te lo creas sin tu ayuda todo esto hubiera sido otro cantar.

También quiero dejar un hueco, para esos amigos a los que nunca me cansaré de ver, amigos del pueblo, de toda la vida; Chacón, López, Chule, García, León, Jesús, Edu, Javi, Miguel, Ruben, etc.. personas con las que he vivido muchas historias y a los que llevo muy dentro.

Para acabar, quiero agradecer a mi familia, de la cual estoy totalmente orgulloso. Mi madre y mi hermana, por darme tanto y pedirme tan poco y a las cuales quiero muchísimo y mi padre, porque con él me tengo que quitar el sombrero. Nunca pensé que alguien me enseñaría tanto.

Tan sólo me queda una persona a la que agradecer todo esto. Esa persona eres tú, Laura. Gracias de todo corazón por toda la ayuda desinteresada, cariño y aguante que has tenido en estos últimos meses. De repente, has conseguido que todo empiece a tener sentido. Siéntete bien orgullosa de ser como eres porque nunca en mi vida he conocido a una persona con tantas virtudes como tú y nunca pierdas esa sonrisa. Gracias, enana, contigo todo es muy fácil.

A todos, muchas gracias.

David Fernández Moreno

Noviembre 2006

Índice

PARTE II ESTADO DE LA CUESTIÓN

PARTE V ANEXOS

Índice de figuras

Índice de tablas

Índice de programas

Acrónimos y abreviaturas

ACL Listas de control de Acceso. Más información en la página 51

ASP *Active Server Pages*. Páginas Activas en el Servidor. Más información en la página 10

API *Application Programming Interface*. Más información en la página 10

CGI *Common Gateway Interface*. Más información en la página 10

CSS *Cascading Style Sheets*. Hojas de estilo en cascada. Más información en la página 14

COCOMO *COnstructive COst MOdel*. Más información en la página 82

DOM *Document Object Model*. Modelo de Objetos del Documento. Más información en la página 14

ECMA *the European Computer Manufacturers' Association*. Más información en la página 14

ESO Enseñanza Secundaria Obligatorio. Más información en la página 4

FI Facultad de Informática. Más información en la página 3

IBM *International Business Machines*. Más información en la página 3

PEAR *Repositorio de Aplicaciones y Extensiones de PHP*.

PECL *PHP Extension Community Library*. Más información en la página 15

PHP *Hypertext Preprocessor*. Preprocesador de hipertexto. Más información en la página 15

PFC *PHP Foundation Classes*. Más información en la página 15

JSP *JavaServer Pages*. Más información en la página 10

JVM *Java Virtual Machine*. Máquina Virtual Java. Más información en la página 10

HTML *Hyper Text Markup Language*. Lenguaje de marcado de hipertexto. Más información en la página 10

HTTP *HyperText Transfer Protocol*. Protocolo de transferencia de hipertextoMás información en la página 15

LDD Lenguaje de definición de datos . Más información en la página 22

LMD Lenguaje de manejo de datos . Más información en la página 22

LOCE Ley Orgánica 10/2002 de 23 de diciembre de Calidad de la Educación.Más información en la página 4

LOE Ley Orgánica de Educación.Más información en la página 4

LOGSE Ley Orgánica 1/1990 de 3 de octubre de Ordenación General del Sistema Educativo.Más información en la página 4

OOWS *Object Oriented Web Solution*. Solución Web Orientada a Objetos. Más información en la página 10

SEQUEL *Structured English Query Language*. Más información en la página 26

SOAP *Simple Object Access Protocol* . Más información en la página 15

SQL *Structured Query Language* . Lenguaje de Consulta Estructurado. Más información en la página 26

SGBD *Sistema Gestor de Bases de Datos*. DataBase Management System. Más información en la página 22

UML *Unified Modeling Lenguage*. Lenguaje Unificado de Modelado. Más información en la página 40

UPM Universidad Politécnica de Madrid. Más información en la página 3

W3C *World Wide Web Consortium*. Más información en la página 14

XML *eXtensible Markup Language*. Lenguaje de Marcas Extensible. Más información en la página 15

Parte I

Introducción y objetivos

Capítulo 1

Introducción

Dentro del proceso de promoción institucional de la UPM, se realizan visitas informativas a distintos centros educativos. El proyecto surge a partir de la necesidad de gestionar estas visitas realizadas por el personal de la Universidad Politécnica de Madrid (UPM).

1.1. Universidad Politécnica de Madrid (UPM)

Aunque la mayoría de sus centros son más que centenarios, la UPM cumplió 25 años en 1996 como tal. Debido a que la mayoría de sus centros se fundaron en los siglos XVIII Y XIX, no es exagerado afirmar que una gran parte de la historia de la tecnología española ha sido desarrollada por sus escuelas, quienes fueron durante muchos años las únicas en la materia.

De todos los estudios que hoy forman la UPM, los primeros en iniciar su andadura fueron los de arquitectura, mientras que uno de los centros más jóvenes es la Facultad de Informática.

El Instituto de Informática de Madrid nació en 1969, fuera del marco universitario, hasta que en 1976 las enseñanzas de informática pasaron a la Universidad y, a la vez, se creó la Facultad de Informática de Madrid, que desde su fundación en Octubre 1977, está integrada en la UPM. Desde el año 1988 se encuentra emplazada en el campus de Montegancedo.

La Facultad de Informática (FI) fue la primera Facultad de Informática de España, junto a las de la Universidad Politécnica de Cataluña y a la Universidad del País Vasco. Durante todos estos años la Facultad ha ido asentando su prestigio, del cual se han beneficiado los ingenieros formados en el centro hasta llegar a ser el centro universitario con mayor experiencia y prestigio en la enseñanza de la Ingeniería Informática superior en Madrid.

Las titulaciones que se imparten son las siguientes:

- Ingeniero en Informática (desde 1996)
- Doctor en Informática
- Máster en Ingeniería del Conocimiento
- Máster en Ingeniería del Software
- Máster Europeo en Computación Lógica
- Máster en Tecnologías de la Información

1.2. Sistema educativo español

El actual sistema educativo queda regulado por la Ley Orgánica 10/2002 de 23 de diciembre de Calidad de la Educación (LOCE), que define la estructura del sistema educativo en sus diversos niveles y etapas. Esta ley presenta novedades respecto a la legislación anterior: Ley Orgánica 1/1990 de 3 de octubre de Ordenación General del Sistema Educativo (LOGSE).

Hoy en día, el Ministerio de Educación y Ciencia, ha presentado el Anteproyecto de una nueva ley llamada Ley Orgánica de Educación (LOE).

El sistema educativo actual definido en la LOCE comprende:

- Educación preescolar (hasta 3 años).
- Enseñanzas escolares (de los 3 a los 18 años).
- Enseñanza universitaria (a partir de los 18 años).

Las enseñanzas escolares pueden ser de régimen general o de régimen especial. Las Enseñanzas escolares de régimen general, se organizan en los siguientes niveles:

- Educación Infantil (3-6 años).
- Educación Primaria (6 - 12 años).
- Educación Secundaria, que comprende las etapas
  - Enseñanza Secundaria Obligatorio. (ESO) (12 - 16 años)
  - Bachillerato (16 - 18 años)
  - Formación Profesional de Grado Medio
- Formación Profesional de Grado Superior.

Por su parte, las enseñanzas escolares de régimen especial son:

- Enseñanzas Artísticas.
- Enseñanzas de Idiomas.
- Enseñanzas Deportivas.

Las visitas que se realizarán en este proyecto se centraran en la enseñanzas de bachillerato, formación profesional de grado medio y formación profesional de grado superior.

1.3. Programa de promoción institucional

En el año 2003, la UPM inició una campaña en centros de enseñanza secundaria dirigida a informar a los potenciales alumnos, acerca de las posibilidades de formación y futuro profesional que les ofrece la universidad.

Una de las acciones de la campaña, fue diseñar un programa de visitas de profesores personalmente a los colegios e institutos de la Comunidad de Madrid. Durante las visitas se expone, en aproximadamente media hora, un resumen de las titulaciones de la UPM, y se responde a las preguntas que los alumnos deseen hacer. En todos los colegios e institutos se entregó [1] diversa documentación.

Desde su implantación, se realizan visitas a unos 150 centros por año. También se realizaron visitas a centros de otras provincias, como Segovia, Guadalajara o Ciudad Real, que así lo solicitaron, llegando a más de 5000 estudiantes de 1º y 2º de bachillerato principalmente y, en menor proporción, también de la ESO.

Igualmente, se realizaron visitas a varios ayuntamientos de la Comunidad de Madrid, que así lo solicitaron a través de sus concejalías.

1.4. Objetivos

El principal objetivo de este trabajo es desarrollar un sistema que facilite la gestión a las visitas del personal de la UPM a los diferentes centros de la Comunidad de Madrid.

Adicionalmente, se desea conseguir:

- Un sistema que sea fácilmente ampliable, es decir, que se puedan añadir fácilmente nuevas funcionalidades

[1] Una guía de la UPM, documentación con los planes de estudio de todas las titulaciones y cuadernillos individualizados con los planes de estudio de cada titulación.

- La posibilidad de que el sistema sea escalable, que pueda soportar más usuarios en el futuro, facultades,..
- Independiente del gestor de base de datos por si se necesita reemplazarlo
- Gestión rápida y eficiente de los recursos.
- Realizar un sistema multiplataforma ,siguiendo los estándares.
- Interfaz vía web amigable para el usuario (*user frienly*)
- Fácil de instalar y mantener
- Ámpliamente documentado

1.5. Contenidos

Este trabajo está divido en dos grandes bloques, en el primero de ellos se va a dar una visión de todas las herramientas, lenguajes y conceptos que se han utilizado para llevar a cabo este proyecto. El segundo bloque va a tratar la parte de implementación del sistema.

En cuanto al primer bloque, se comenzará describiendo la arquitectura empleada y el conjunto de lenguajes que se han utilizado para realizar la aplicación. Seguidamente, se estudiará la gestión de la base de datos, detallando el sistema gestor utilizado, su seguridad, etc. por último, se da una visión global del proceso de desarrollo seguido para su obtención incidiendo principalmente en los ciclos de vida y metodologías empleadas.

El segundo bloque comenzará analizando los requisitos del sistema y sus restricciones. Después, se describirá con detalle el diseño utilizado para realizar el sistema y, a continuación, se detallarán aspectos curiosos de la implementación de la aplicación.

Para finalizar, se expondrá las conclusiones, valoraciones y experiencia adquirida durante la realización del proyecto.

Parte II

Estado de la cuestión

Capítulo 2

Aplicaciones Web

Habitualmente, las aplicaciones muestran datos y ejecutan en la misma máquina. Sin embargo, en los últimos años, se ha popularizado el desarrollo de aplicaciones con interfaz web, esto es, controladas desde el navegador. Esto facilita el acceso a las mismas desde cualquier lugar con conexión a la red. Además, la mejora de las conexiones facilita su implantación.

2.1. Arquitectura de aplicaciones web

El término arquitectura de aplicación es usado en el diseño de aplicaciones, habitualmente del tipo cliente-servidor.

En el diseño físico se especifica exactamente donde se encontraran las piezas de la aplicación. En el diseño lógico o conceptual se especifica la estructura de la aplicación y sus componentes sin tomar en cuenta donde se localizará el software, hardware e infraestructura. Tales conceptos incluyen el orden de procesamiento, mantenimiento y seguimiento comunes en sistemas organizacionales.

Muchas veces se toma demasiado en cuenta el diseño físico de una aplicación. Por añadidura los desarrolladores generalmente asumen, indebidamente, que el diseño lógico se corresponde punto a punto con el diseño físico. Contrario a esto, un diseño adecuado debería permitir su implantación en varias plataformas y configuraciones. Como se puede ver, esta característica de portabilidad es un punto deseable para permitir que su aplicación sea flexible y escalable.

2.1.1. Modelo cliente / servidor

El paradigma cliente / servidor es uno de los más extendidos dentro de los servicios a través de red. Este término, en su más amplia definición, se usa para describir una aplicación en la cual dos o

más procesos separados trabajan juntos para completar una tarea. El proceso cliente solicita al proceso servidor la ejecución de alguna acción en particular. Esta operación se conoce como proceso cooperativo, dado que dos procesos separados cooperan para completar la tarea en particular.

Figura 2.1: Modelo cliente / servidor

La Figura 2.1 muestra el funcionamiento de este modelo. Se puede ver como el cliente realiza peticiones al servidor, mientras que el servidor se dedica simplemente a responderle. De por si, el servidor tiene un papel pasivo por lo que necesita que un cliente le demande algo. Los principales servicios de Internet (WWW, FTP, SMTP, etc.) tienen clientes y servidores específicos, aunque en tiempos recientes se intente integrar todo bajo una interfaz web que es más amigable para el usuario.

Los procesos pueden o no estar en una sola máquina física. Tales procesos en una aplicación cliente / servidor pueden localizarse en la misma máquina o separados físicamente. El diseño lógico, y no el físico, es el que determina en que grado una aplicación es cliente / servidor.

2.1.2. Aplicaciones web

La idea fundamental de los navegadores, *browsers*, es presentar documentos escritos en HTML que han obtenido de un servidor web. Estos documentos HTML habitualmente presentan información de forma estática, sin más posibilidad de interacción con ellos.

El modo de crear los documentos HTML ha variado a lo largo de la corta vida de las tecnologías web pasando desde las primeras páginas escritas en HTML almacenadas en un fichero en el servidor web hasta aquellas que se generan al vuelo como respuesta a una acción del cliente y cuyo contenido varia según las circunstancias.

Además, el modo de generar páginas dinámicas ha evolucionado, desde la utilización del *Common Gateway Interface* (CGI), hasta los *servlets* pasando por tecnologías tipo *JavaServer Pages* (JSP). Todas

estas tecnologías se encuadran dentro de aquellas conocidas como *Server Side*, ya que se ejecutan en el servidor web.

Con la extensión de Internet y de la web en concreto, se han abierto infinidad de posibilidades en cuanto al acceso a la información desde casi cualquier sitio. Esto representa un desafío a los desarrolladores de aplicaciones, ya que los avances en tecnología demandan cada vez aplicaciones más rápidas, ligeras y robustas que permitan utilizar la web.

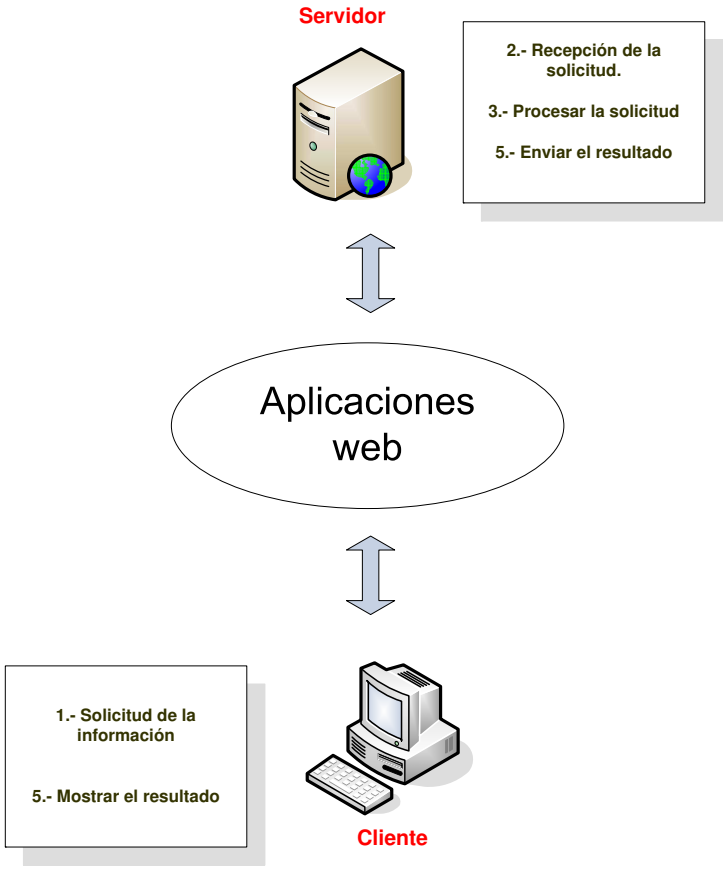


Figura 2.2: Pasos en el modelo cliente / servidor

Afortunadamente, se dispone de herramientas potentes para alcanzar este objetivo, ya que han surgido nuevas técnicas que permiten que el acceso a una base de datos desde la web. El único problema es decidir entre el conjunto de posibilidades la correcta para cada situación.

El protocolo CGI ha cumplido con el propósito de añadir interactividad a las páginas web pero sus deficiencias en el desarrollo de aplicaciones y en la escalabilidad de las mismas ha conducido al desarrollo de nuevas *Application Programming Interface* (API) específicas de servidor. Para aprovechar el potencial de estas tecnologías y ofertar una solución de servidor más extensible y portable, *Sun* posee la tecnología llamada *servlet*. Los *servlets Java* son eficientes, debido al esquema de *threads* en el que se basan y al

uso de una arquitectura estándar como la *Java Virtual Machine* (JVM).

Otra nueva tecnología viene a sumarse a las que extienden la funcionalidad de los servidores web, llamada JSP. Los JSP permiten unir HTML, aplicaciones Java, y componentes *JavaBeans* creando una página web especial que el servidor web compila dinámicamente en un *servlet* la primera vez que es llamada.

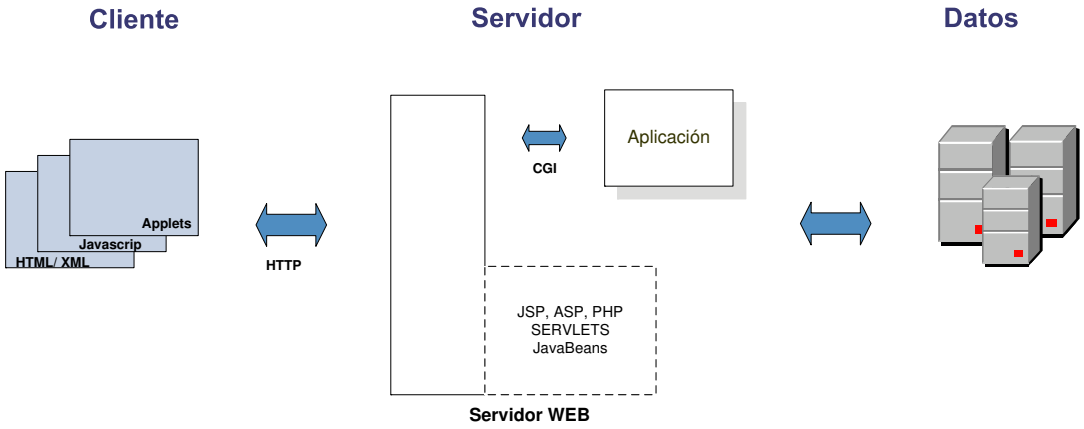


Figura 2.3: Esquema general de las tecnologías Web

Otro aspecto que completa el panorama son, las inclusiones del lado del cliente, *Client Side*, que se refieren a las posibilidades de que las páginas lleven incrustado código que se ejecuta en el cliente, como por ejemplo *JavaScript*.

El esquema general de la situación se puede ver en la Figura 2.3, donde se muestran cada tipo de tecnología involucrada en la generación e interacción de documentos web.

2.1.3. Arquitectura de 3 niveles

La llamada arquitectura en 3 niveles es la más común en sistemas de información que, además de tener una interfaz de usuario, contemplan la persistencia de los datos.

Una descripción de los tres niveles seria la siguiente:

Nivel 1: Presentación ventanas, informes, etc.

Nivel 2: Lógica de la Aplicación. tareas y reglas que gobiernan el proceso.

Nivel 3: Almacenamiento mecanismo de almacenamiento.

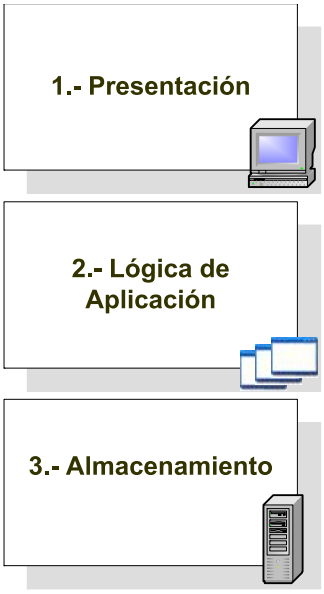


Figura 2.4: Arquitectura de 3 niveles

2.2. Lenguajes

Para la creación de aplicaciones web se utilizan múltiples lenguajes. En este caso, se han utilizado diferntes lenguajes de programación web como pueden ser HTML, PHP o JavaScript

2.2.1. *Hyper Text Markup Language* (HTML)

El *Hyper Text Markup Language* (HTML) [Specification, 2004], es un lenguaje de marcado, diseñado para estructurar textos y definir su presentación en forma de hipertexto, que es el formato estándar de las páginas web. Gracias a Internet y a los navegadores del tipo *Mozilla, Firefox, Netscape o Explorer*, el HTML se ha convertido en uno de los formatos más populares que existen para la construcción de documentos.

Contrariamente a otros lenguajes de programación, el HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma con la que deben aparecer el texto, así como las imágenes y los demás elementos, en la pantalla del ordenador.

2.2.2. *Cascading Style Sheets* (CSS)

Las hojas CSS son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML . El *World Wide Web Consortium* (W3C) [Consortium, 2004], es el encargado de formular la especificación de las hojas de estilo que servirá de estándar para los navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. Por ejemplo, el elemento de HTML `H1` indica que un bloque de texto es un encabezamiento y que es más importante que un bloque etiquetado como `H2`. Cuando se utiliza CSS, la etiqueta `H1` no debería proporcionar información sobre como va a ser visualizado, solamente marca la estructura del documento. La información de presentación se proporciona separada en una hoja de estilo con la que se especifica como se ha de mostrar: color, fuente, alineación del texto y tamaño, además puede ser adjuntada tanto como un documento separado o en el mismo documento HTML.

Las ventajas de utilizar CSS son:

- Control centralizado de la presentación de un sitio web completo, con lo que se agiliza de forma considerable la actualización del mismo.
- Los navegadores permiten a los usuarios especificar su propia hoja de estilo local, que será aplicada a un sitio web remoto, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales podrían configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.
- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa o mostrada en un dispositivo móvil.
- El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

Hay varias versiones: CSS1 y CSS2, con CSS3 en desarrollo por el W3C. Los navegadores modernos implementan CSS1 de forma completa, aunque existen pequeñas diferencias de implementación dependiendo del navegador.

2.2.3. Javascript

JavaScript [Eich, 2001], es un lenguaje interpretado orientado a las páginas web basado en el paradigma prototipo, con una sintaxis semejante a la del lenguaje Java.

El lenguaje Javascript se integra dentro del código HTML de las páginas web y actúa en cuanto un evento (un click en un botón, por ejemplo) es ejecutado.

El lenguaje que fue inventado por Brendan Eich en la empresa *Netscape Communications*, apareció por primera vez en el *Netscape Navigator 2.0*. Tradicionalmente, se venía utilizando en páginas web HTML, para realizar tareas y operaciones en el marco de la aplicación cliente / servidor.

Figura 2.5: JavaScript

Los autores inicialmente lo llamaron *Mocha* y, más tarde, *LiveScript*, pero fue rebautizado como *JavaScript* en un anuncio conjunto entre *Sun Microsystems* y *Netscape*, en 1995.

En 1997, los autores propusieron *JavaScript* para que fuera adoptado como estándar internacional *the European Computer Manufacturers' Association*. (ECMA). En junio de 1997, fue adoptado como un estándar ECMA, con el nombre de *ECMAScript*. Poco después también lo fue como un estándar ISO.

Para evitar estas incompatibilidades, el *World Wide Web Consortium* (W3C) diseñó el estándar *Document Object Model* (DOM), que incorporan los navegadores actuales.

2.2.4. *Hypertext Preprocessor* (PHP)

PHP [PHP, 2004], es un lenguaje *Open Source* interpretado de alto nivel, especialmente pensado para desarrollos web usado para generar páginas HTML. La mayoría de su sintaxis es similar a C, Java y *Perl* y es fácil de aprender. La meta de este lenguaje es permitir el desarrollo de páginas web, páginas dinámicas de una manera rápida y fácil, aunque su versatilidad hace que pueda emplearse en otro muchos ámbitos.

Figura 2.6: PHP

En el invierno de 1998, poco después del lanzamiento oficial de PHP 3.0, Andi Gutmans y Zeev Suraski comenzaron a trabajar en la reescritura del núcleo de PHP. Los objetivos de diseño fueron mejorar la ejecución de aplicaciones complejas, y la modularidad del código.

El nuevo motor, apodado *Motor Zend* [Zend, 1999] (comprimido de sus apellidos, Zeev y Andi), alcanzó estos objetivos de diseño satisfactoriamente, y se introdujo por primera vez a mediados de 1999.

PHP 4.0, basado en este motor, fue oficialmente liberado en mayo de 2000. Esta versión incluye otras características clave como el soporte para la mayoría de los servidores web y sesiones *HyperText Transfer Protocol* (HTTP).

Hoy en día, se estima que PHP es usado por cientos de miles de programadores y muchos millones de sitios informan que lo tienen instalado, sumando más del 20 % de los dominios en Internet. Además su equipo de desarrollo incluye docenas de programadores, trabajando en el núcleo o en proyectos relacionados con PHP como PEAR y el proyecto de documentación.

Actualmente tiene soporte para cosas tan dispares como *Simple Object Access Protocol* (SOAP), *eXtensible Markup Language* (XML), *Sockets*, *DB, OpenSSL* o llamadas *POSIX*.

2.2.4.1. *Repositorio de Aplicaciones y Extensiones de PHP* (PEAR)

PEAR [PEAR, 2004a], es un repositorio muy completo de clases en PHP, en el cual se puede encontrar desde clases para manejar ecuaciones matemáticas o para generar gráficos, hasta clases para generar hojas de calculo en *Excel*, de una forma fácil y con funciones pocas veces vista con PHP. PEAR esta dividido en dos partes principales: PEAR y PCL, el primero lo constituyen paquetes escritos completamente en PHP, el segundo esta compuesto por clases escritas en C o C++.

Figura 2.7: Pear

El propósito de PEAR es proveer:

- Una biblioteca *Open Source* estructurada para los usuarios de PHP
- Un sistema para distribución de código y mantenimiento de paquetes
- Una normativa de estilo para código escrito en PHP
- Las clases de la fundación PHP (*PHP Foundation Classes* (PFC)).
- La biblioteca de extensión (*PHP Extension Community Library* (PECL)).
- Una comunidad de usuarios

PEAR surgió hace más de 5 años, la primera clase que se publicó fue DB, la cual hoy en día es la clase oficial de PHP para abstracción de base de datos, es decir, para poder fácilmente cambiar de gestor de base de datos cambiando tan solo unas líneas de código.

A partir de la versión 4.3.0, el paquete principal de PEAR junto con los paquetes más comunes, se instalan automáticamente, cuando se realiza la instalación de PHP, en las versiones anteriores hay que instalarlo por separado. Cuando ya se tiene el paquete principal instalado, los demás paquetes se agregan de forma muy sencilla. Hay que tomar en cuenta dependencias entre paquetes, por ello habrá que instalar dichos paquetes con anticipación.

2.2.4.2. Plantillas *Smarty*

Smarty [SMARTY, 2004],es un motor de plantillas para PHP, que facilita la separación entre la lógica de la aplicación y el contenido.

Figura 2.8: Plantilla Smarty

En la mayoría de los casos y en proyectos de gran envergadura el programador de la aplicación y el diseñador de la plantilla no son la misma persona. Por ejemplo: se crea una página web de un artículo de un diario. En ella, el encabezado del articulo, el rótulo, el autor y el cuerpo son elementos del contenido, que no contienen información de como van a ser presentados. Estos se pasan por la aplicación *Smarty*, donde el diseñador crea la plantilla, y usa una combinación de etiquetas HTML y etiquetas de plantilla para formatear la presentación de estos elementos (HTML, tablas, color de fondo, tamaño de letras, hojas de estilo, etc.). Si un día el programador necesita cambiar la manera de recuperar el contenido del articulo, este cambio no afectará al diseñador de la plantilla, ya que el contenido llegara a la plantilla exactamente igual. De la misma manera, si el diseñador de la plantilla quiere re-diseñarla en su totalidad, estos cambios no afectaran a la lógica de la aplicación.

Por lo tanto, el programador puede hacer cambios en la lógica de la aplicación sin que sea necesario reestructurar la plantilla. Y el diseñador de la plantilla puede hacer cambios sin que afecte a la lógica.

Smarty lee la plantilla y crea los *scripts* de PHP. Estos *scripts* son ejecutados, y por consiguiente no existe ningún costo por analizar cada archivo de *template*.

Algunas de las características de *Smarty*:

- Rápido y eficiente.
- No analiza gramaticalmente desde arriba el *template*, sólo lo procesa una vez.
- Sólo recompila los archivos de plantilla que fueron cambiados.

- Puede crear funciones comunes, de modo que el lenguaje de la platilla es altamente extensible.
- Sintaxis de etiquetas delimitadoras configurables.
- Los constructores *if, elseif, else, endif* son procesados por el interprete de PHP, así la sintaxis de la expresión *if* ... puede ser compleja o simple de la forma que se quiera.
- Permite un anidamiento ilimitado de *sections, ifs, etc*.
- Es posible incrustar directamente código PHP en los archivos de plantilla.
- Fuentes de plantilla absoluto
- Funciones habituales de manipulación de cache
- Arquitectura de *plugin*

2.2.5. Seguridad

En los últimos años, se han desarrollado multitud de aplicaciones web que acceden a bases de datos, estas aplicaciones, al estar disponibles a través de la web, son más propensas a recibir ataques que permitan acceder o modificar la base de datos, es por ello que el tema de seguridad es de especial interés.

2.2.5.1. Autenticación

Un sistema de autenticación es un módulo de seguridad para asegurar que el usuario que visita las páginas es quien dice ser. Por supuesto, conociendo a ese usuario, se le podrá dar acceso a más aspectos de la página que si fuese un usuario desconocido o anónimo.

En la Figura 2.9 se muestra el proceso de autenticación, donde se pide un usuario y una contraseña para acceder a la aplicación de acceso restringido.

Una vez que se tienen los datos de autenticación (usuario y contraseña escritos en la página inicial), se hace una comprobación de los mismos y, se redirecciona al navegador a la página de la aplicación restringida, en caso de que sean correctos, o a la página del login donde se volverá a pedir el usuario y la contraseña, en caso de que sean incorrectos.

La aplicación de acceso restringido, aparte de mostrar las funcionalidades que se quieren proteger con usuario y contraseña, debe de realizar unas comprobaciones de seguridad para saber si se ha pasado con éxito el proceso de autenticación o si se está intentando acceder de manera no permitida a esa página.

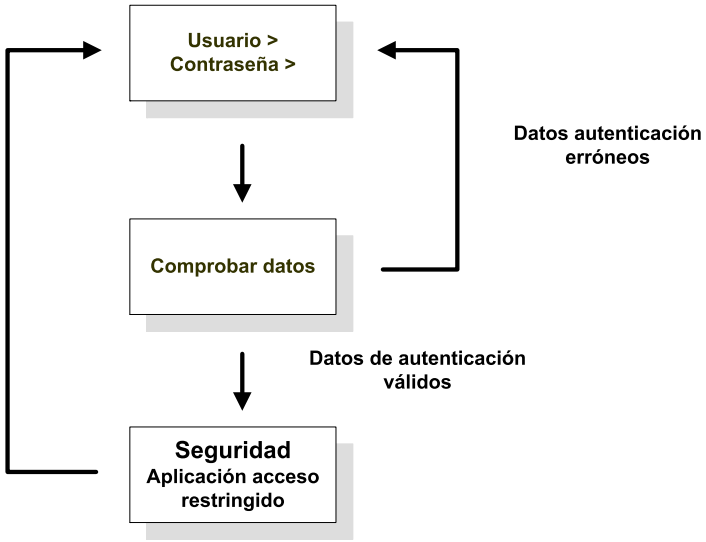


Figura 2.9: Autenticación del usuario

2.2.5.2. Ataques típicos

De todos los ataques, existen dos que son especialmente peligrosos: la inyección de SQL y la suplantación de identidad.

2.2.5.2.1. Suplantación de indentidad Para evitar la repetición de la contraseña, se almacenan las credenciales en la sesión del usuario en forma de token (espacio de almacenamiento temporal en el servidor). A partir de ese momento, el navegador manda un identificador de sesión (en la URL o como *cookie*) para usarla.

Este ataque consiste en robar la sesión de un usuario autenticado, es decir, ese identificador y usarlo para acceder a la aplicación con las credenciales del usuario.

2.2.5.2.2. Inyección de código SQL La inyección SQL consiste en la modificación del comportamiento de las consultas mediante la introducción de parámetros no deseados en los campos a los que tiene acceso el usuario.

Este tipo de errores puede permitir a usuarios malintencionados acceder a datos a los que de otro modo no tendrían acceso y, en el peor de los casos, modificar el comportamiento de las aplicaciones.

El acceso restringido se basa en una tabla de usuarios y contraseñas y sólo los usuarios que se validen contra esa tabla podrán acceder a esos contenidos. Una manera de que los usuarios se validen será mostrar un formulario pidiendo el usuario y la contraseña y una vez que se rellenan esos campos enviar esos datos a la base de datos para comprobar si es válido.

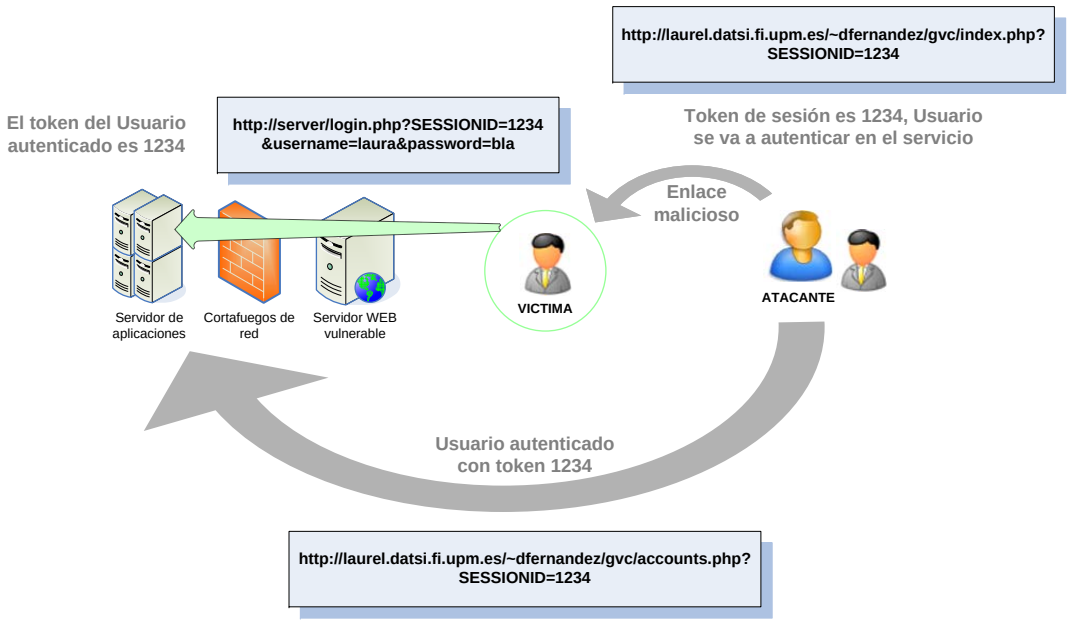


Figura 2.10: Suplantación de indentidad

Si el usuario escribe por ejemplo *admin* y de contraseña cualquier otra cosa, la sentencia no devolverá registros y no se permitirá entrar a esa persona. Pero ¿qué ocurre si el usuario escribe ' or '1'='1 como usuario y lo mismo de contraseña? En este caso la variable Consulta contendrá la cadena:

```
SELECT Count(*) FROM Usuarios WHERE Usuario = '' or '1'='1' AND password = '' or '1'='1'
```

Y obviamente esta sentencia devuelve registros con lo que el usuario entrará en la web sin tener permiso.

Pero es aún peor si el usuario utiliza estos mecanismos de inyección de SQL para ejecutar código arbitrario en el servidor: sentencias DDL, cambio de permisos, utilizar procedimientos almacenados y un largo etcétera.

Como solución, hay varios sistemas para evitarlo. Por ejemplo se puede filtrar las entradas de los usuarios reemplazando la aparición ' de " por (dos comillas simples) e incluso evitando que los usuarios puedan pasar caracteres como ' / o cualquier otro que se nos ocurra que puede causar problemas.

Otro factor importante en cuanto a la seguridad es limitar al máximo los permisos del usuario que ejecuta estas sentencias para evitar posibles problemas. Por ejemplo utilizando un usuario distinto para las sentencias *SELECT, DELETE, UPDATE* y asegurándonos que cada ejecución de una sentencia ejecute una sentencia del tipo permitido.

Capítulo 3

Bases de datos

El término base de datos fue acuñado por primera vez en 1963, en un simposio celebrado en California. De forma sencilla, una base de datos no es más que un conjunto de información relacionada que se encuentra agrupada de forma estructurada.

El archivo por sí mismo, no constituye una base de datos, sino más bien la forma en que está organizada la información es la que da origen a la base de datos. Las bases de datos manuales pueden ser difíciles de gestionar y modificar. Por ejemplo, en una guía de teléfonos no es posible encontrar el número de un individuo si no se sabe su apellido, aunque se conozca su domicilio. Del mismo modo, en un archivo de pacientes en el que la información está ordenada por el nombre, será una tarea complicada encontrar todos los pacientes que viven en una zona determinada. Todos estos problemas se pueden resolver mediante el uso de una base de datos informatizada.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos. Desde el punto de vista más formal, se puede definir una base de datos como un conjunto de datos estructurados, fiables y homogéneos, organizados independientemente en máquina, accesibles en tiempo real, y que son compartidos por usuarios concurrentes que tienen necesidades de información diferente y no predecibles en el tiempo.

Éstas colecciones de datos que cumplen las siguientes propiedades:

1. Están estructurados independientemente de las aplicaciones y del soporte de almacenamiento que los contiene.

2. Presentan la menor redundancia posible.

3. Pueden ser compartidos por varios usuarios y/o aplicaciones.

3.1. *Sistema Gestor de Bases de Datos* (SGBD)

Los sistemas gestores de bases de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos (almacenamiento físico en disco) y el usuario, o las aplicaciones que la utilizan. Se compone de:

Lenguaje de definición de datos. Una vez finalizado el diseño de una base de datos y escogido un SGBD para su implementación, el primer paso consiste en especificar el esquema conceptual, el esquema interno de la base de datos, y la correspondencia entre ambos. En muchos SGBD no se mantiene una separación estricta de niveles, por lo que el administrador de la base de datos y los diseñadores utilizan el mismo lenguaje para definir ambos esquemas: el Lenguaje de definición de datos (LDD). El SGBD posee un compilador de LDD cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos de los esquemas y almacenar la descripción del esquema en el catálogo o diccionario de datos. Se dice que el diccionario contiene metadatos: describe los objetos de la base de datos.

Cuando en un SGBD hay una clara separación entre los niveles conceptual e interno, el LDD sólo sirve para especificar el esquema conceptual. Para especificar el esquema interno se utiliza un lenguaje de definición de almacenamiento (LDA). Las correspondencias entre ambos esquemas se pueden especificar en cualquiera de los dos lenguajes. Para tener una verdadera arquitectura de tres niveles sería necesario disponer de un tercer lenguaje, el lenguaje de definición de vistas (LDV), que se utilizaría para especificar las vistas de los usuarios y su correspondencia con el esquema conceptual.

Lenguaje de manipulación de datos . Una vez creados los esquemas de la base de datos, los usuarios necesitan un lenguaje que les permita manipular los datos de la base de datos: realizar consultas, inserciones, eliminaciones y modificaciones. Este lenguaje es el que se denomina LMD.

Hay dos tipos de Lenguaje de manejo de datos (LMD):

- Procedurales
- No Procedurales

Con un LMD procedural el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo hay que obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. De esta manera, se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados.

Un LMD no procedural se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones del LMD desde un terminal o bien desde en un lenguaje de programación de alto nivel. Los LMD no procedurales permiten especificar los datos a obtener en una consulta o los datos que se deben actualizar, mediante una sola y sencilla sentencia. El usuario o programador especifica qué datos quiere obtener sin decir cómo se debe acceder a ellos. El SGBD traduce las sentencias del LMD en uno o varios procedimientos que manipulan los conjuntos de registros necesarios. Esto libera al usuario de tener que conocer cuál es la estructura física de los datos y qué algoritmos se deben utilizar para acceder a ellos.

Lenguaje de consulta. Los sistemas de bases de datos disponen de unos lenguajes: conjunto de instrucciones que de acuerdo a una sintaxis ayudan a realizar las distintas funciones que ha de cumplir un SGBD. El usuario final no necesita normalmente tanta potencia, por ello se le da un lenguaje de manipulación con sintaxis sencilla, a veces por medio de menús. La estructura y sintaxis de todos esos tipos de lenguajes dependen de cada SGBD, pero en las SGBD relacionales, SQL es un estándar muy extendido.

E. Codd, el creador del modelo relacional, ha establecido una lista con los ocho servicios que debe ofrecer todo SGBD.

1. Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento).

2. Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena:

   - Nombre, tipo y tamaño de los datos.
   - Nombre de las relaciones entre los datos.
   - Restricciones de integridad sobre los datos.
   - Permisos de acceso a la base de datos.
   - Esquemas externos, conceptual e interno, y correspondencia entre los esquemas.
   - Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.
- Se puede tener un historial de los cambios realizados sobre la base de datos.
- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.
- Se puede establecer políticas de la seguridad.
- Se puede garantizar la integridad.
- Se puede proporcionar información para auditorías.

3. Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización (debido, por ejemplo, a un fallo del hardware) la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.

4. Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pue-

den interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

5. Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado previo consistente.

6. Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

7. Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea viable.

8. Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar.

Además, de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

1. Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil.

2. Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

   - Herramientas para importar y exportar datos.
   - Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
   - Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
   - Herramientas para reorganización de índices.
   - Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

3.1.1. *Structured Query Language* (SQL)

La historia de SQL [Chamberlin, 1996], empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de *International Business Machines* (IBM), de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional.

Este lenguaje se llamaba *Structured English Query Language* (SEQUEL) cuyo primer prototipo, llamado *SEQUEL-XRM*, fue implementado entre 1974 y 1975. Los experimentos con ese prototipo condujeron, en los años siguientes a una revisión del lenguaje *(SEQUEL/2)*, que a partir de ese momento cambio de nombre por motivos legales, convirtiéndose en SQL.

El prototipo *(System R)*, basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL.

A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo *Oracle y Sybase*) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL como estándar para los lenguajes relacionales y en 1987 se transformo en estándar ISO. Esta versión del estándar se denominó SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido, primero, a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la ínter comunicabilidad entre todos los productos que se basan en él. Efectivamente, en general cada productor adopta e implementa en la propia base de datos solo el corazón del lenguaje SQL (el así llamado *Entry level* o al máximo el *Intermediate level*), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, esta en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva versión de SQL deberían ser su transformación en un lenguaje *standalone* y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimedia.

3.1.2. SQlite

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, y que está contenida en una relativamente pequeña librería en C. *SQLite* es un proyecto de dominio público creado por D. Richard Hipp.

A diferencia de los sistemas de gestión de base de datos cliente-servidor, el motor de *SQLite* no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la librería *SQLite* se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de *SQLite* a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar accesible en el sistema de ficheros.

La librería implementa la mayor parte del estándar SQL92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento, y durabilidad (ACID), *triggers* y la mayor parte de las consultas complejas.

SQLite usa un sistema de tipos inusual. En lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un *string* en una columna de tipo entero (a pesar de que *SQLite* tratará en primera instancia de convertir la cadena en un entero). Algunos usuarios consideran esto como una innovación que hace que la base de datos se mucho más útil, sobretodo al ser utilizada desde un lenguaje de *scripting* de tipos dinámicos. Otros usuarios lo ven como un gran inconveniente, ya que la técnica no es portable a otras bases de datos SQL.

Figura 3.1: SQLite

3.2. Modelado de datos

De acuerdo a Ullman [Ullman, 1999], un modelo de datos es un sistema formal y abstracto que permite describir los datos de acuerdo con reglas y convenios predefinidos. Es formal pues los objetos del sistema se manipulan siguiendo reglas perfectamente definidas y utilizando exclusivamente los operadores definidos en el sistema, independientemente de lo que estos objetos y operadores puedan significar.

Segun Silberschatz [Silberschatz, 1998], un modelo de datos es una combinación de tres componentes:

1. Una colección de estructuras de datos, los bloques constructores de cualquier base de datos que conforman el modelo.

2. Una colección de operadores o reglas de inferencia, los cuales pueden ser aplicados a cualquier instancia de los tipos de datos listados en el punto anterior, que permiten consultar o derivar datos de cualquier parte de estas estructuras en cualquier combinación deseada

3. Una colección de reglas generales de integridad, las cuales, explícita o implícitamente, definen un conjunto de estados consistentes estas reglas algunas veces son expresadas como reglas de insertar-actualizar-borrar

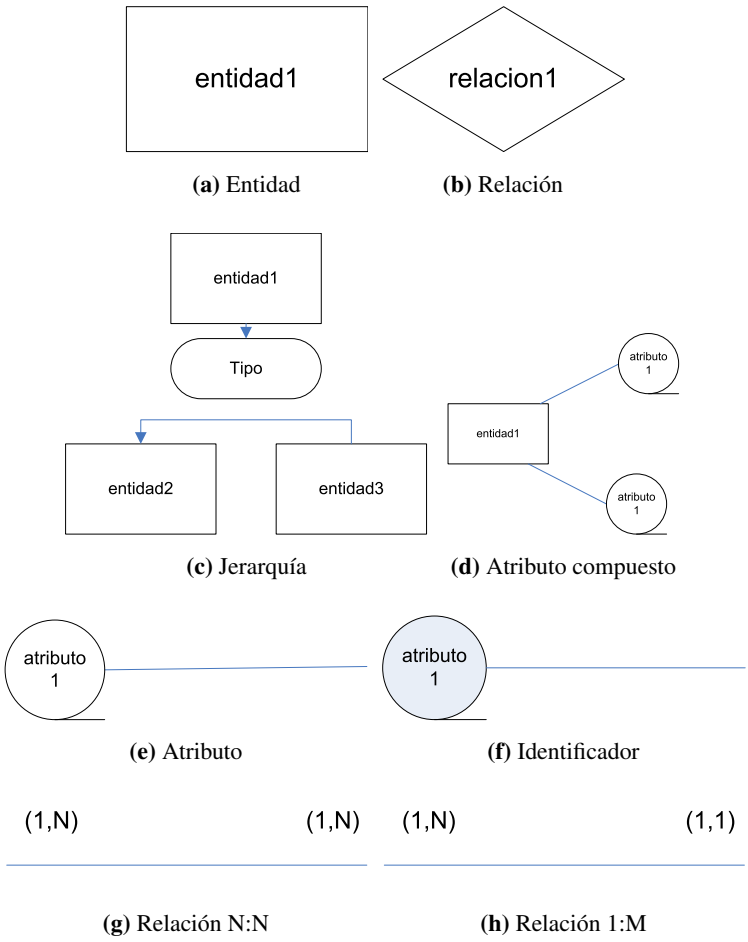
En paralelo al diseño del sistema se ha desarrollado el modelo de base de datos.

3.2.1. Modelo entidad / relación

Los diagramas entidad / relación [Chen, 1976], a veces denominado por su siglas E-R, son una herramienta para el modelado de datos de un sistema de información. Estos diagramas expresan entidades relevantes para un sistema de información, sus interrelaciones y propiedades.

Los diagramas E-R son un lenguaje gráfico para describir conceptos. Informalmente, son simples dibujos o gráficos que describen la información que trata un sistema de información y el software que lo automatiza.

Los elementos de dicho lenguaje (Figura 3.2) se describen a continuación, por orden de importancia:



(a) Entidad **(b)** Relación

(c) Jerarquía **(d)** Atributo compuesto

(e) Atributo **(f)** Identificador

(g) Relación N:N **(h)** Relación 1:M

Figura 3.2: Elementos de modelo entidad / relación

Entidades: Una entidad es cualquier objeto discreto sobre el que se tiene información. Se representa mediante un rectángulo o caja etiquetada en su interior mediante un nombre (Figura 3.2a). Ejemplos de entidades habituales en los sistemas de información son: factura, persona, albarán, empleado, etc.

Relaciones: Una relación describe cierta interdependencia (de cualquier tipo) entre una o más entidades. Se representa mediante un rombo etiquetado en su interior mediante un verbo (Figura 3.2b). Además, dicho rombo debe unirse mediante líneas con las entidades que relaciona. Una relación no tiene sentido sin las entidades que relaciona. Por ejemplos una persona (entidad) trabaja (relación) para un departamento (entidad).

Las relaciones entre dos entidades se denominan binarias, las relaciones entre tres entidades se denominan ternarias, y las relaciones entre cuatro o más entidades se denominan múltiples. Las relaciones múltiples son poco frecuentes, mientras que las relaciones binarias son habituales en

cualquier problema.

También son posibles las relaciones reflexivas donde una entidad se relaciona consigo misma. Esto significa que una instancia de una entidad se relaciona con otra instancia distinta de la misma entidad. Por ejemplo: una persona (entidad) contrae matrimonio (relación) con otra persona (la misma entidad).

Atributos: Los atributos son propiedades relevantes propias de una entidad. Se representan mediante un círculo o elipse etiquetado mediante un nombre en su interior (Figura 3.2e). Cuando un atributo es identificativo de la entidad se suele subrayar dicha etiqueta.

Por motivos de legibilidad, los atributos no suelen representarse en un diagrama entidad-relación, sino que se describen textualmente en otros documentos adjuntos.

Los atributos describen información útil sobre las entidades. En particular, los atributos identificativos son aquellos que permiten diferenciar a una instancia de la entidad de otra distinta. Por ejemplo, el atributo identificativo que distingue a un empleado de otro es su número de la Seguridad Social.

El modelado de datos no acaba con el uso de esta técnica. Son necesarias otras técnicas para lograr un modelo directamente implementable en una base de datos.

- Simplificación de relaciones múltiples en binarias, o ternarias en los casos apropiados.
- Normalización de relaciones (algunas relaciones pueden transformarse en atributos y viceversa).
- Conversión a tablas (en caso de utilizar una base de datos relacional. Este caso se verá más detenidamente en el siguiente punto).

3.2.2. Paso a tablas

Todo diagrama E-R se representa mediante una colección de tablas para su implementación en el gestor apropiado.

Para cada una de las entidades y relaciones existe una tabla única a la que se le asigna como nombre el del conjunto de entidades y de las relaciones respectivamente, cada tabla tiene un número de columnas que son definidas por la cantidad de atributos y las cuales tienen el nombre del atributo.

Representación tabular de los conjuntos de entidades fuertes. Sea E un conjunto de entidades fuertes con los atributos descriptivos $\{a_1, a_2, \cdots, a_n\}$. Esta entidad se representa mediante una tabla llamada E con n columnas distintas, cada una de las cuales corresponde a uno de los atributos de E. Cada fila de la tabla corresponde a una entidad del conjunto de entidades E.

Se puede añadir una nueva entidad a la base de datos insertando una fila en una tabla. También se pueden borrar o modificar las filas.

Representación tabular de los conjuntos de entidades débiles. Sea A un conjunto de entidades débiles con los atributos $\{a_1, a_2, \cdots, a_m\}$. Sea B el conjunto de entidades fuertes del que A depende. Sea la clave primaria de B el conjunto de atributos $\{b_1, b_2, \cdots, b_m\}$. Se representa el conjunto de entidades A mediante una tabla llamada A con una columna por cada uno de los atributos del conjunto: $\{a_1, a_2, \cdots, a_m\} \cup \{b_1, b_2, \cdots, b_m\}$.

Representación tabular de los conjuntos de relaciones. Sea R un conjunto de relaciones, sean $\{a_1, a_2, \cdots, a_m\}$ el conjunto de atributos formados por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R, y sean $\{b_1, b_2, \cdots, b_m\}$ los atributos descriptivos de R (si los hay). El conjunto de relaciones se representa mediante una tabla llamada R con una columna por cada uno de los atributos del conjunto: $\{a_1, a_2, \cdots, a_m\} \cup \{b_1, b_2, \cdots, b_m\}$.

Para cada conjunto de entidades de la base de datos y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene varias columnas, cada una de las cuales tiene un nombre único. Las restricciones especificadas en un diagrama E-R, tales como las claves primarias y las restricciones de cardinalidad, se corresponden con restricciones sobre las tablas generadas a partir del diagrama E-R.

Las restricciones especificadas en un diagrama E-R, tales como las claves primarias y las restricciones de cardinalidad, se corresponden con restricciones sobre las tablas generadas a partir del diagrama E-R.

3.3. Seguridad en la base de datos

Las bases de datos normalmente son accedidas por varios usuarios por lo que el SGBD debe proporcionar técnicas que permitan restringir a algunos usuarios o grupos de usuarios el acceso a determinadas partes de la base de datos. Esto es especialmente importante cuando se trata de organizaciones que cuentan con una gran base de datos que almacena los datos de distintas secciones de la organización. Para ello, los SGBD suelen contar con un subsistema de seguridad y autorización que asegura la seguridad de la base de datos frente accesos no autorizados. Estos mecanismos de seguridad suelen ser de dos tipos:

Mecanismos de seguridad discrecionales. Permiten conceder privilegios a los usuarios de la base de datos.

Mecanismos de seguridad obligatorios. Permiten definir varios niveles de seguridad para los objetos de la base de datos, de forma que los usuarios pueden utilizar sólo los elementos que están en su nivel de seguridad o inferior.

Otro problema de seguridad es el de evitar que personas no autorizadas tengan acceso al SGBD. El mecanismo de seguridad de un SGBD debe incluir formas que permitan restringir el acceso al sistema, ésa función se denomina control de acceso y está basado en el uso de cuentas de usuario.

El administrador de la base de datos (DBA) es la persona responsable de la base de datos y entre sus obligaciones se encuentra la de conceder privilegios a los usuarios que necesitan utilizar el sistema. El DBA tiene una cuenta de superusuario que ofrece privilegios no disponibles para las cuentas convencionales. Entre estos privilegios se encuentran el de creación de cuentas, concesión y revocación de privilegios, y asignación de niveles de seguridad.

De esta forma, todas las personas que deseen acceder a la base de datos, tendrán que solicitar al DBA la creación de una cuenta de usuario con un nombre de usuario y una contraseña. El usuario utilizará estos datos y podrá acceder a la base de datos siempre que el SGBD valide dicha conexión. El SGBD podrá guardar todas las operaciones que se realicen sobre la base de datos, de forma que se puedan detectar anomalías en el acceso o en la modificación de los datos. Estas operaciones son guardadas en el registro del sistema *(log)* y el número de entradas correspondientes a operaciones realizadas sobre la base de datos dependerá del grado de detalle con el que se quieran registrar las operaciones sobre la base de datos.

Capítulo 4

Proceso de desarrollo

En algunos casos la generación de una aplicación software consiste en la codificación del programa y, en casos más avanzados, un pequeño documento resumiendo la funcionalidad del código. Esta falta de formalización del proceso produce sistemas poco robustos y difíciles de mantener. Según se incrementa el tamaño del programa, al añadir nueva funcionalidad, resulta cada vez más complicado comprender el funcionamiento del sistema, debido a las múltiples interrelaciones existentes entre los distintos componentes que quedan además completamente ocultas en el código.

El problema se agrava cuando en el desarrollo del sistema interviene un grupo de personas o es necesario que alguien utilice la funcionalidad implementada. Al no existir documentación suficiente sobre el proceso de desarrollo del sistema, son necesarias reuniones para aclarar la funcionalidad y la aparición de continuas inconsistencias que ocasionan grandes pérdidas de tiempo. En estas condiciones es posible que la realización del sistema sea inviable. Para evitar estos problemas es necesario seguir un proceso de desarrollo en el cual se definan los hitos intermedios que deben alcanzarse hasta la obtención del sistema total. Según se avanza en el proceso se obtiene el sistema y un conjunto de productos auxiliares (habitualmente documentación) que permiten definir de forma completa dicho sistema, permitiendo a terceras personas comprender su funcionamiento y, por tanto, facilitar la inclusión de nuevos componentes en el grupo de desarrollo o que terceros puedan utilizar el software desarrollado.

Asimismo se facilita el establecimiento de puntos de control para verificar la corrección del sistema en cada hito y tomar las acciones correctoras oportunas lo antes posible.

En la Figura 4.1 se muestra una visión general del proceso de desarrollo. En primer lugar, están las necesidades, expresadas como requisitos del sistema, es decir, el problema que el sistema debe resolver, por otro lado, está toda la lógica de la aplicación, es decir, todo el mecanismo que se va a llevar a cabo para realizar el proyecto. En esta parte se utilizarán una serie de tecnologías y lenguajes.

Por último ,se tendrá el resultado final que, en este caso, es una aplicación vía web, la cual estará preparada para su utilización.

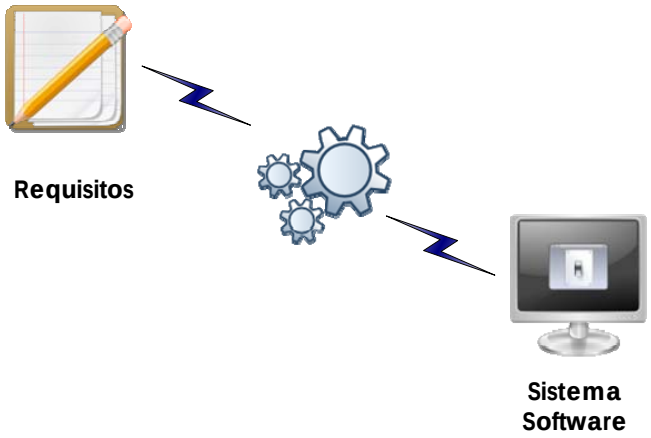


Figura 4.1: Elementos de la aplicación

4.1. Ciclo de vida

El ciclo de vida determina las fases que deben llevarse a cabo para la realización del proyecto desde que se plantea el problema (habitualmente por parte del cliente) hasta que es resuelto por un sistema informático. Los ciclos de vida suelen asociarse al proceso de desarrollo software pero, debido a la evolución de los sistemas informáticos, actualmente tienden a aplicarse a todo el proceso del desarrollo informático. Existen múltiples ciclos de vida que abarcan diversos problemas según sus características. Por ello también existen diversas clasificaciones, siendo una de las posibles la clasificación por el grado de incertidumbre que admiten.

Figura 4.2: Clasificación de la incertidumbre

4.1.1. Ciclos de vida de referencia

En la Figura 4.2, se plantean los ciclos de vida más usuales que mejor se adaptan a las características del sistema a desarrollar. Estos ciclos de vida se estructuran en diversas fases que pueden usarse de referencia para posteriores desarrollos, aunque es necesario tener un conocimiento suficiente de cada uno de ellos para poder utilizarlos de forma adecuada.

4.1.1.1. Secuencial o en cascada

Se trata de la primera formalización del proceso de desarrollo software propuesta por Royce [Royce, 1970], introduciendo una cierta disciplina en la ingeniería del software que permite corregir algunas deficiencias como la pronta codificación sin análisis y diseño.

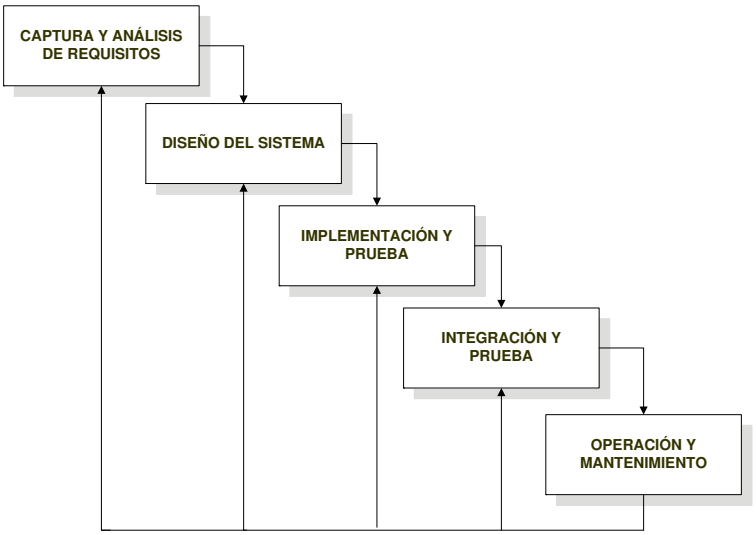


Figura 4.3: Ciclo de vida secuencial

Modela el ciclo de vida clásico (ingeniería del sistema, análisis, diseño, codificación, pruebas y mantenimiento) con un enfoque secuencial en el que cada fase es completamente finalizada antes de comenzar la siguiente. Esto es posible únicamente si el sistema a desarrollar esta fijo desde el inicio, es decir, los requisitos son estables.

Debido a la evolución de las tecnologías y el mayor tamaño de los proyectos junto a la aparición de requisitos variables, se introduce una re-alimentación al finalizar cada fase que palia levemente los problemas del ciclo de vida en estas circunstancias. A pesar de la re-alimentación, el ciclo de vida es bastante inflexible, siendo muy complicado introducir cambios de forma sencilla en el sistema.

4.1.1.2. Prototipado

El prototipado [Gomma, 1995], surge como respuesta a los requisitos variables producidos por la existencia de partes mal definidas o mal comprendidas. Para esas partes se construye un prototipo, que puede ser en papel o computadora, centrado en la interfaz hombre-sistema, en alguna funcionalidad o bien puede simular una parte del sistema real para comprobar su adecuación para el usuario.

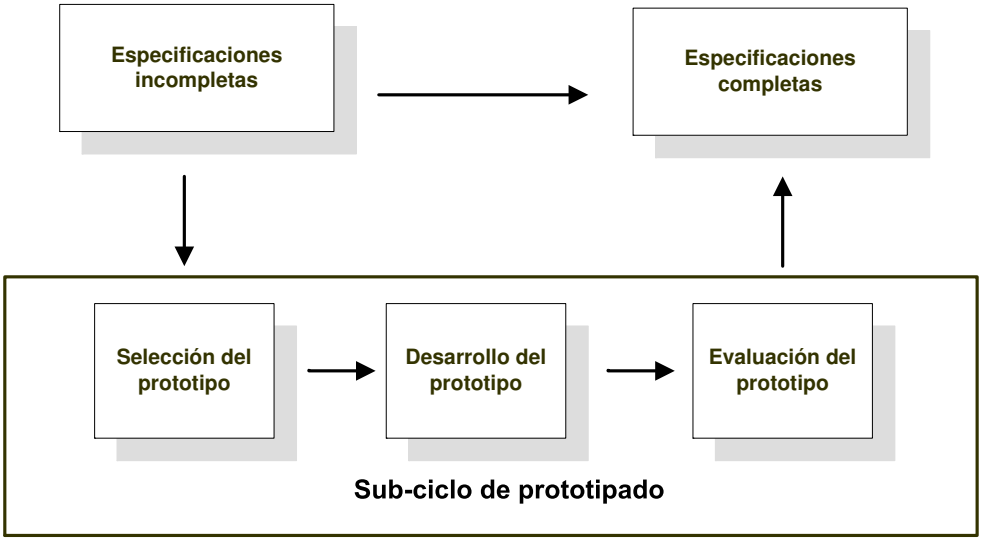


Figura 4.4: Ciclo de vida prototipado

Una variación de este ciclo de vida es el prototipado rápido, en el cual se crea un primer prototipo que es utilizado en las primeras fases para analizar las necesidades del cliente y extraer de esta forma los requisitos de partes no definidas. Es deseable poder evolucionar este primer prototipo para generar el sistema final, aunque no siempre es posible o rentable.

El uso de prototipos se ha incluido como una fase más en otros ciclos de vida. Esta técnica, como herramienta de análisis, es muy aconsejable para evitar el rechazo del cliente o cuando la interfaz de usuario sea una parte muy importante del sistema.

4.1.1.3. Espiral

El ciclo de vida en espiral [Boehm, 1981], se basa en sucesivos ciclos de experimentación y aprendizaje, a través de los cuales se incrementa la funcionalidad, dando lugar a sucesivas versiones del software cada vez más completas.

La primera iteración se centra en la parte fundamental del sistema y las sucesivas iteraciones construyen el sistema, sobre la base formada por los ciclos previos, de forma incremental.

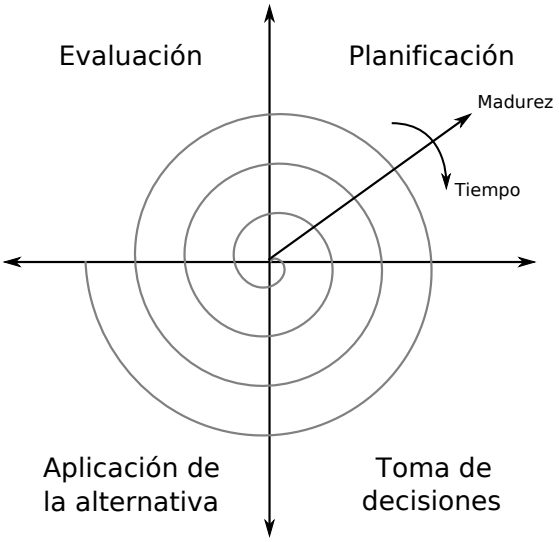


Figura 4.5: Ciclo de vida en espiral

En cada ciclo se plantean cuatro fases (planificación, análisis de riesgos, ingeniería y evaluación) que recubren e incrementan las tradicionales. Al incluir en cada ciclo el análisis de riesgos y la evaluación junto al desarrollo incremental, es posible responder a los distintos riesgos que plantea la existencia de incertidumbre en los desarrollos tomando las acciones correctoras necesarias.

Basándose en este esquema de sucesivas iteraciones surgen dos modelos:

Incremental: Debido a las necesidades variables, el problema no se trata como un todo, sino que se realiza el desarrollo completo de un subconjunto y seguidamente se construye sobre este otro subconjunto, incrementando así el sistema.

Evolutivo: Al igual que el incremental, se plantea un desarrollo por fases, pero en este caso variando lo realizado, mientras que crece poco a poco hasta lograr el sistema requerido. Su mayor inconveniente es que se plantea desde un inicio la modificación de lo realizado en las primeras fases. El ciclo de vida evolutivo permite un desarrollo más flexible que el incremental, puesto que permite realizar una búsqueda de la solución retractándose (modificando) cuando se detecta una gran desviación entre el sistema desarrollado y el requerido.

Un enfoque erróneo del desarrollo en espiral es considerar que cada ciclo equivale a una cascada. La cascada avanza todo el desarrollo, por lo que es imposible realizar ajustes correctivos puesto que ya ha sido desarrollada toda la fase para todo el sistema. La espiral por su parte solo realiza una parte del sistema, que puede incluso no ser ninguna actividad de ingeniería de software como aprender nuevas técnicas.

4.1.2. Ciclo de vida empleado

Debido a las características del sistema, se plantea, por lo tanto, un ciclo de vida con incertidumbre media y, más concretamente, el ciclo de vida incremental, con el que se creará el sistema partiendo de sus piezas base hasta obtener el sistema completo.

Al plantear el desarrollo de esta manera, es posible admitir cierto grado de incertidumbre, facilitando la absorción de pequeños cambios en el mismo como se verà más adelante en la descripción del desarrollo.

Los principales riesgos a priori del sistema son:

- Desconocimiento de las nuevas técnicas, lo que conlleva un periodo de aprendizaje y adaptación.
- Posibles variaciones en las especificaciones, que repercutirán en el resto del sistema

4.2. Metodología orientada a objetos

Cuando se va a construir un sistema software es necesario conocer un lenguaje de programación, pero con eso no basta. Si se quiere que el sistema sea robusto y mantenible es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto, que incluya las actividades principales. Si se sigue un proceso de desarrollo que se ocupa de plantear cómo se realiza el análisis y el diseño, y cómo se relacionan los productos de ambos, entonces la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente, especialmente cuando se trata de un equipo de desarrollo formado por varias personas.

Para este proyecto se va a utilizar el método de desarrollo orientado a objetos que propone Craig Larman [Larman, 1999]. Este proceso no fija una metodología estricta, sino que define una serie de actividades que pueden realizarse en cada fase, las cuales deben adaptarse según las condiciones del proyecto que se esté llevando a cabo. Las fases que se plantean son:

1. Planificación y especificación de requisitos
2. Construcción
    - Diseño software
    - Implementación
    - Pruebas

3. Instalación

La notación que se usa para los distintos modelos, tal y como se ha dicho anteriormente, es la proporcionada por UML, que se ha convertido en el estándar de facto en cuanto a notación orientada a objetos. El uso de UML permite integrar con mayor facilidad en el equipo de desarrollo a nuevos miembros y compartir con otros equipos la documentación, pues es de esperar que cualquier desarrollador versado en orientación a objetos conozca y use UML (o se esté planteando su uso).

El enfoque que toma es el de un ciclo de vida iterativo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido a la interfaz hombre - máquina. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

4.2.1. Diseño orientado a objetos

El proceso a seguir para realizar desarrollo orientado a objetos es complejo. Este proceso está formado por una serie de actividades y sub-actividades, cuya realización se va repitiendo en el tiempo aplicadas a distintos elementos.

En este apartado se va a presentar una visión general para poder tener una idea del proceso a alto nivel. Las tres fases al nivel más alto son las siguientes:

Planificación y Especificación de Requisitos: Planificación, definición de requisitos, construcción de prototipos.

Construcción: La construcción del sistema. Las fases dentro de esta etapa son las siguientes:

- *Diseño de Alto Nivel:* Se analiza el problema a resolver desde la perspectiva de los usuarios y de las entidades externas que van a solicitar servicios al sistema.
- Diseño de Bajo Nivel: El sistema se especifica en detalle, describiendo cómo va a funcionar internamente para satisfacer lo especificado en el Diseño de Alto Nivel.
- *Implementación:* Se lleva lo especificado en el Diseño de Bajo Nivel a un lenguaje de programación.
- *Pruebas:* Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado.

Instalación: La puesta en marcha del sistema en el entorno previsto de uso.

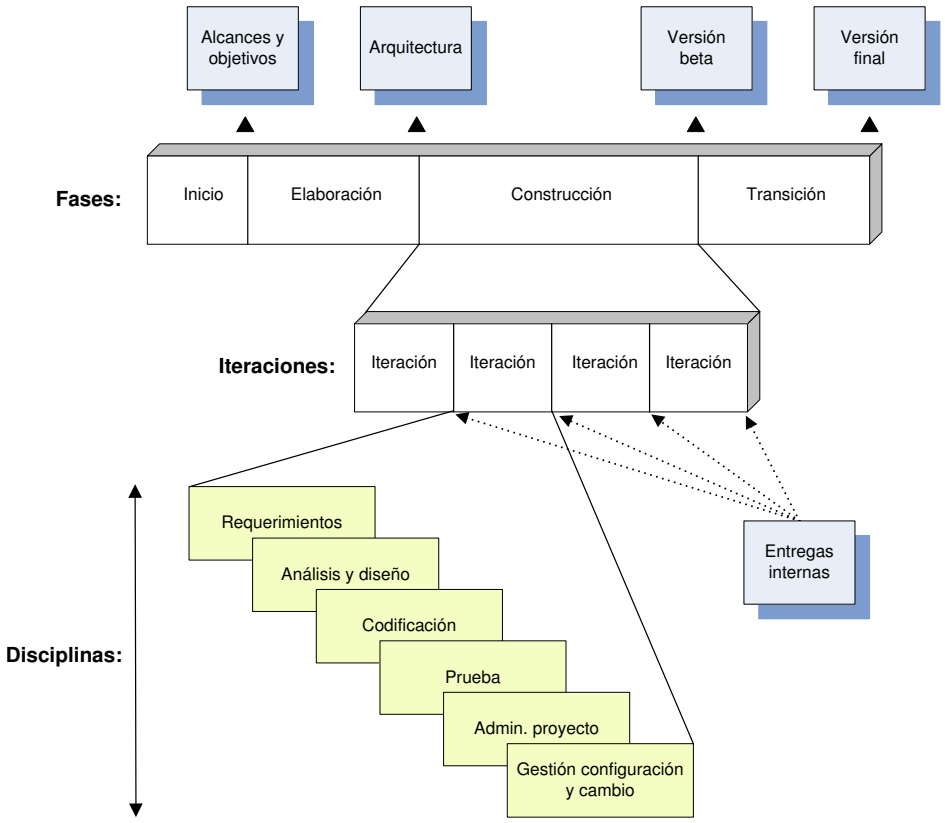


Figura 4.6: Fases del Proceso Unificado

De ellas, la fase de construcción es la que va a consumir la mayor parte del esfuerzo y del tiempo en un proyecto de desarrollo. Para llevarla a cabo se va adoptar un enfoque iterativo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del diseño de alto y bajo nivel hasta la implementación y pruebas, tal y como se muestra en la Figura 4.6.

El sistema va creciendo incrementalmente en cada ciclo. Con esta aproximación se consigue disminuir el grado de complejidad que se trata en cada ciclo, y se tiene pronto en el proceso una parte del sistema funcionando que se puede contrastar con el usuario/cliente.

4.2.2. *Unified Modeling Lenguage* (UML)

UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos ([Ferré Grau, 2003]). Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa *Rational Software Co*, para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de

gran peso en la industria como *Microsoft, Hewlett-Packard, Oracle o IBM*, así como grupos de analistas y desarrolladores.

Esta notación ha sido ámpliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (principalmente Booch,OMT y OOSE). UML ha puesto fin a las llamadas guerras de métodos que se han mantenido a lo largo de la década de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

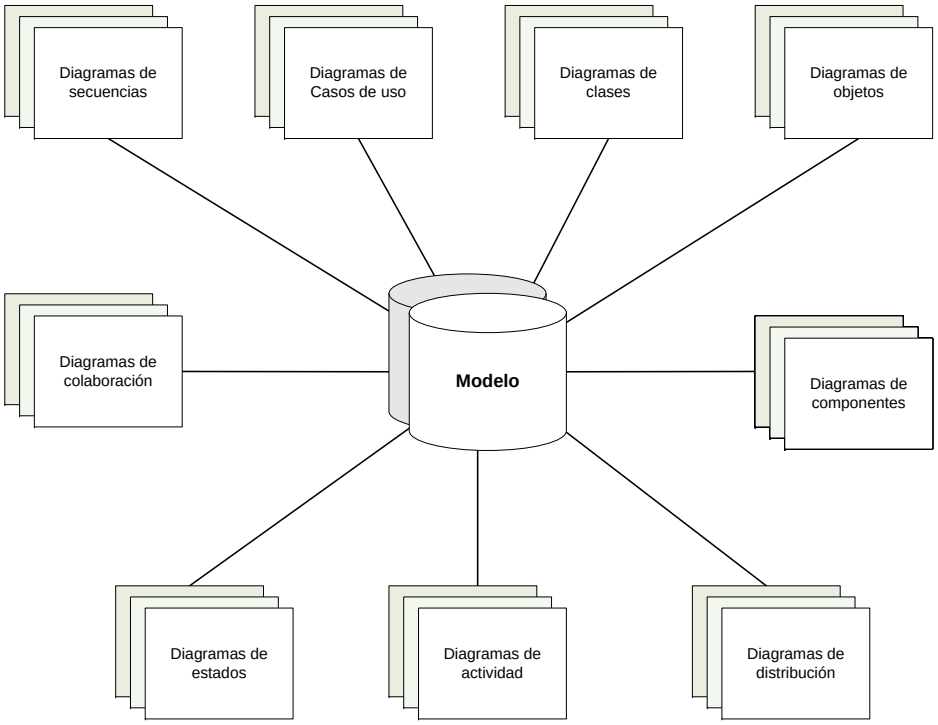


Figura 4.7: Modelado con UML

Hay que destacar que UML solamente es una notación para sistemas orientados a objetos que facilita la comunicación e integración de diversos grupos de desarrollo. No esta ligado a ningún ciclo de vida o lenguaje de desarrollo.

4.2.3. *Object Oriented Web Solution* (OOWS)

Hoy en día, con la rápida expansión de Internet y los avances en el área de las tecnologías web, han surgido un nuevo tipo de aplicaciones en estos entornos, cada vez más complejas y dinámicas, comúnmente conocidas como *aplicaciones web*. Además, debido al acelerado crecimiento y la alta com-

petitividad de las actividades comerciales en la Red, estos sistemas son construidos en periodos de tiempo muy cortos, sin el apoyo de herramientas de trabajo adecuadas y utilizando soluciones *ad-hoc*, lo que está llevando a construir sistemas software de baja calidad y de difícil mantenimiento y evolución.

En los últimos años han surgido gran cantidad de aproximaciones metodológicas (OOHDM, WebML, UWE, WSDM, AutoWeb, etc.) que intentan ayudar en la sistematización de la construcción de soluciones en ambientes Web, proporcionando mecanismos de abstracción que faciliten la conceptualización y el desarrollo de estos sistemas. Estos mecanismos permiten especificar características hipermediales, (*'hiper' + media'*, es decir, navegación + multimedia) y funcionales, dando soporte a la gestión de usuarios (diferentes tipos, adaptación, personalización, etc.). Además, se están intentando definir marcos de trabajo integrados que proporcionen herramientas adecuadas para dar soporte a la construcción de estos sistemas en todas sus fases. Como resultado de estos trabajos, surge lo que también hoy en día está acuñado como *Ingeniería web*.

La intención es definir un método de desarrollo que permita especificar sistemas software para ambientes web que capture las necesidades de estos sistemas web, mediante especificaciones conceptuales y permita obtener prototipos operacionales a partir de estas especificaciones de manera automática.

Para conseguirlo, se ha extendido un método orietnado a objetos existente, *OO-Method*, que permite capturar las propiedades funcionales del sistema que se consideran relevantes para construir una especificación textual OO y formal de manera automática. En este contexto se han concentrado muchos esfuerzos hacia el desarrollo de nuevos modelos para enriquecer este método de producción de software orientado a objetos con la expresividad necesaria para especificar características navegacionales y de presentación de información orientadas a las aplicaciones web.

Estas especificaciones formales constituyen un repositorio de información de alto nivel del sistema que será utilizado como entrada a un compilador de modelos conceptuales, que utilizando una estrategia de traducción basada en patrones (de la especificación a la implementación), hacen posible la construcción de una implementación operacional, generando un prototipo del sistema completo (incluyendo características estáticas y dinámicas) en la plataforma destino del sistema. Esta extensión del método *OO-Method* con capacidades navegacionales y de presentación es lo que llamamos OOWS.

4.2.4. Modelo de Navegación

En este punto, se deben captar los requisitos de navegación de la aplicación. Para ello, se introduce el modelo de navegación que permite dar una vista navegacional (mapa navegacional) sobre el diagrama de clases de *OO-Method* para cada tipo de usuario que pueda interactuar con el sistema.

Esta vista proporcionará una estructura de accesibilidad (definiendo el conjunto posible de caminos

de navegación) en función del tipo de usuario. Así la semántica navegacional de las aplicaciones web se captura en función de cada agente del sistema identificado, permitiendo una personalización a nivel de modelado conceptual del acceso en función de las necesidades de cada tipo de usuario. Para construir este modelo, se realizan 2 tareas:

1. *Clasificación e Identificación de usuarios.* Se realiza un estudio de los diferentes potenciales tipos de usuario que pueden interactuar con el sistema y sus interrelaciones.

2. *Construcción de los mapas navegacionales.* Para cada usuario detectado, se construye su vista navegacional del sistema (basado en el modelo del dominio definido previamente).

4.2.5. Modelo de Presentación

Una vez definido el modelo de navegación es necesario especificar las características de presentación del sistema. Para este propósito se introduce el modelo de presentación, que complementa la información capturada en el modelo de navegación. Este modelo utiliza los nodos o contextos navegacionales como entidades básicas para definir las propiedades de presentación de información.

Parte III

Implementación del problema

Capítulo 5

Análisis de requisitos del sistema

El desarrollo de cualquier sistema software no consiste únicamente en la codificación del programa, aunque sea una de las fases mas importantes, sino que existen múltiples tareas que deben ser desarrolladas paralelamente si se desea un sistema robusto y de fácil mantenimiento y con garantías de finalización en el tiempo y coste estimados. Para determinar las tareas que deben de ser llevadas a cabo es necesario seguir un proceso de desarrollo software, adaptándolo según se estime conveniente al proyecto en el cual se aplique.

Una de estas tareas es la documentación. La documentación asociada al proyecto debe permitir conocer en todo momento el estado en el que se encuentra el sistema pero su generación no debe ser una sobrecarga innecesaria durante el desarrollo.

El primero de los documentos a desarrollar es la especificación de requisitos software. Este documento tiene como objetivo definir, de forma clara y no ambigua, las características y cualidades del sistema software a desarrollar, siendo el punto de partida del proyecto y el contrato con el cliente. El documento redactado para tal propósito sigue una normativa estándar [IEEE 830.1998].

Como se indicó en la Sección 4.1, se ha empleado un ciclo en espiral realizando varias iteraciones. En alguna de ellas los requisitos fueron alterados por lo que se expone el conjunto final de requisitos tras todas las iteraciones realizadas.

Los requisitos se han dividido en bloques semánticos: Funcionalidad, requisitos tecnológicos y restricciones del sistema. Cada requisito es numerado para facilitar la trazabilidad.

5.1. Funcionalidades

5.1.1. Autenticación

Requisito 5.1.1

Autenticar (log in)

El sistema deberá pedir autenticación. Se validará el acceso mediante el usuario y la contraseña. En el caso de que un usuario sea completamente nuevo para el sistema debe registrarse antes de acceder a las funcionalidades del mismo.

Si un usuario se autentica correctamente, abre una sesión con el sistema. Una vez autenticado aparecerá una pantalla con todas las operaciones disponibles. Dependiendo de los roles que tenga la persona asignados se mostrarán unas operaciones u otras.

5.1.2. Gestión de usuarios

Requisito 5.1.2

Tipos de Usuarios

En el sistema, van a ver tres tipos de usuarios: el coordinador, el conductor y el profesor. Cada uno de ellos tendrá una serie de funcionalidades que se pueden ver en la Tabla 5.1.

Funcionalidades	**Coordinador**	**Profesor**	**Conductor**
Alta usuario	Si	No	No
Baja usuario	Si	No	No
Ver usuarios	Si	No	No
Alta colegio	Si	No	No
Alta visita	Si	No	No
Alta persona de contacto	Si	No	No
Listar colegios	Si	No	No
Alta visita	Si	No	No
Buscar visita	Si	Si	Si

Tabla 5.1: Funcionalidades asignadas a los roles

Requisito 5.1.3

Registrar usuarios

El sistema debe permitir dar de alta a los usuarios para que estos puedan acceder al sistema. Esta tarea la llevaran a cabo lo coordinadores de las facultades.

Requisito 5.1.4

Modificar usuarios existentes

Los coordinadores del sistemas podrán cambiar cualquier perfil de los usuarios registrados. Un usuario ordinario solo podrá cambiar su contraseña.

Requisito 5.1.5

Eliminar usuarios existentes

Los coordinadores pueden eliminar los perfiles de sus usuarios. Realmente, eliminar el usuario es inhabilitar su acceso al sistema, puesto que la información no se eliminará realmente. El coordinador podrá rehabilitar a un usuario.

Los usuarios no podrán borrar su propio perfil.

Requisito 5.1.6

Listar usuarios

El sistema deberá ser capaz de listar a todos los usuarios de cada facultad. En esa lista se mostrarán tanto los usuarios activos y inactivos.

Requisito 5.1.7

Dar de alta a personas de contacto

Los coordinadores del sistema deben poder dar de alta a personas de contacto para los colegios, cada colegio puede tener mas de una persona de contacto. Una persona de contacto solamente estará asociada a un único colegio.

Requisito 5.1.8

Modificar personas de contacto

Los coordinadores del sistemas podrán modificar los perfiles de las personas asociadas a los colegios.

Requisito 5.1.9

Eliminar personas de contacto

Los coordinadores del sistema podrán dar de baja a personas de contacto de un colegio. Al igual que con los usuarios, no se borra su perfil sino que se desactiva.

5.1.3. Gestión de grupos

Requisito 5.1.10

Dar de alta grupos

Los coordinadores del sistema deben poder dar de alta a grupos de usuarios. Cada grupo que se cree en el sistema solamente podrá estar asociado a una sola facultad.

Requisito 5.1.11

Asignar operaciones a los grupos

Los coordinadores del sistema deberán poder asignar operaciones a grupos, modificar operaciones del grupo y eliminar operaciones el grupo.

Requisito 5.1.12

Modificar grupos

Los coordinadores del sistema podrán modificar los perfiles de sus grupos.

Requisito 5.1.13

Eliminar grupos

Los coordinadores del sistema podrán eliminar grupos del sistema. Este grupo no se borra realmente de la base de datos, lo que ocurre es que se desactiva para que no se muestre en el sistema.

Cuando se borra un grupo no se puede volver a activar, como pasa con los usuarios borrados.

5.1.4. Gestión de colegios

Requisito 5.1.14

Dar de alta colegios

Los coordinadores del sistema deben poder dar de alta a colegios dentro del sistema.

Requisito 5.1.15

Asignar facultades a colegios

Los coordinadores del sistema podrán asignar facultades a colegios, cambiar la facultad asociada al colegios y eliminar la facultad asignada a los colegios [1]. Un colegio solamente estará asociado a una única facultad.

Requisito 5.1.16

Modificar colegios

Los coordinadores del sistema podrán modificar los perfiles que tienes los colegios en el sistema.

Requisito 5.1.17

Eliminar colegios

Los coordinadores del sistema podrán eliminar colegios en el sistema.

Requisito 5.1.18

Buscar colegios

[1] Éste requisito no se encuentra totalmente implementado. Actualmente el sistema asigna facultades a colegios, quedando las siguientes condiciones como líneas futuras

El sistema deberá permitir hacer una búsqueda de colegios por distintos parámetros, tanto a los coordinador como a todos los usuarios que tengan permisos para ello.

Requisito 5.1.19

Listar Colegios

El sistema deberá permitir listar todos los colegios del sistema, tanto a los coordinador como a todos los usuarios que tengan permisos para ello.

5.1.5. Gestión de visitas

Requisito 5.1.20

Dar de alta visitas

Los coordinadores del sistema deben poder dar de visitas a colegios dentro del sistema.

Requisito 5.1.21

Modificar una visita

Los coordinadores del sistema podrán modificar los datos de una visita creada en el sistema

Requisito 5.1.22

Eliminar visitas

Los coordinadores del sistema podrán eliminar una visita en el sistema.

Requisito 5.1.23

Buscar visitas

Los usuarios del sistema podrán buscar las visitas.

5.2. Requisitos tecnológicos

Requisito 5.2.1

Control de Usuarios

El sistema deberá llevar un control de usuarios mediante el mecanismo de ACL.

Requisito 5.2.2

Independencia de la base de datos

No debe haber dependencia de ningún SGBD.

Requisito 5.2.3

Generación de informes

El sistema deberá generar informes sobre visitas y cartas para colegios[2]..

5.3. Restricciones del Sistema

Requisito 5.3.1

Sistema modulable y extensible

El sistema deberá ser modular y fácilmente extensible. La inclusión de nuevas funcionalidades deberá ser sencilla.

Requisito 5.3.2

Rapidez

El sistema deberá ser suficientemente rápido y eficaz.

Requisito 5.3.3

Acceso al sistema

El sistema no permitirá el acceso a las personas de contactos de los distintos colegios.

[2]Éste requisito no se encuentra totalmente implementado. Actualmente el sistema permite rellenar informes y almacenarlos en la base de datos, pero no tiene implementado ningún mecanismo para generarlo, por ejemplo, a PDF.

Capítulo 6

Diseño de la Aplicación

> El propósito del diseño es de crear una arquitectura para la naciente implementación, [...] el diseño arquitectural sólo puede comenzar una vez que el equipo tenga un entendimiento razonable de los requerimientos del sistema. [...] El diseño, como el análisis, nunca termina realmente hasta que el sistema final es entregado. Durante esta fase, se alcanza un cierto grado de culminación al poner en su lugar la mayoría de las decisiones estratégicas de diseño y al establecer políticas para diversos problemas tácticos. [...] El diseño se enfoca en la estructura, estática y dinámica, su propósito principal es de crear el 'esqueleto' concreto del sistema sobre del cual todo el resto de la implementación se basa. [Grady Booch, 1999]

Estas palabras definen claramente qué es el diseño. La creación de la estructura básica del sistema es la tarea clave, aunque también se buscan otras cosas, en particular patrones que simplifiquen el diseño y posibilidades de reuso entre otras.

La parte diseño se ha dividido en dos bloques generales, por un lado está el diseño software y por otro el diseño de datos. En el diseño software se va a comenzar describiendo los casos de uso cuya función es cubrir toda la funcionalidad del sistema y su interacción, para continuar con el modelo del dominio que refleja la identificación de las distintas entidades necesarias, y por último se describirá el modelo conceptual de la aplicación web.

En la parte de diseño de datos, se representa el modelo entidad/relación de la aplicación y su paso a tablas para la base de datos.

6.1. Diseño Software

La exposición del diseño incluye una visión general del sistema para posteriormente detallar los puntos fundamentales del mismo. Como en el caso de lo requisitos, se muestra el diseño alcanzado tras todas las iteraciones realizadas.

En la Figura 6.1 se muestra una visión general por capas de la aplicación. En la parte superior se encuentra la capa de presentación, es decir, la interfaz. Esta capa se subdivide a su vez en dos capas más específicas; el HTML y las plantillas *Smarty*. Mediante estas capas, se generan las distintas páginas web.

Una capa por debajo se encuentra la lógica de la aplicación. En esta capa se incluye toda la funcionalidad de todas las operaciones que soportará el sistema.

Por último se encuentra la capa de abstracción de almacenamiento, que proporciona acceso a la base de datos. Esta capa a su vez tiene subcapas, la primera de ellas; SGBD, es la encargada de definir el lenguaje con el que se van a realizar las consultas a la base de datos, en el caso el SQL, y más concretamente *SQLite*. Seguidamente, hay una subcapa de abstracción de la bases de datos donde se encuentra el PEAR, y que como se puede ver en la figura esta formado por el módulo `DB_TABLE`, el `PEAR_DB` y el `MDB2`.

La idea es tener una capa entre sistema y la base de datos que oculte la complejidad de la misma y toda dependencia con ella. Si se programara usando directamente las sentencias (o funciones) que provee el propio lenguaje, se estaría fuertemente vinculado al gestor de la base de datos.

6.1.1. Casos de uso

Un caso de uso describe a los actores utilizando un sistema para satisfacer un objetivo. Es una historia o una forma particular de usar un sistema. Los casos de uso se corresponden con requisitos, en particular requisitos funcionales.

Nótese que UML no define un formato para describir un caso de uso. Tan solo define la representación gráfica entre actores y funcionalidad en un diagrama: el diagrama de casos de uso. Sin embargo, junto a la representación gráfica del caso de uso, cada uno de ellos dispone de una descripción textual explicativa.

En el diagrama de casos de uso (Figura 6.2) se distinguen los actores (usuarios del sistema):

Coordinador: con privilegios para controlar todas las operaciones del sistema. Esta persona tiene permisos para realizar cualquier operación.

Profesor: encargado de realizar visitas a los distintos centros educativos.

Conductor: encargado de llevar al conductor a las visitas que tenga.

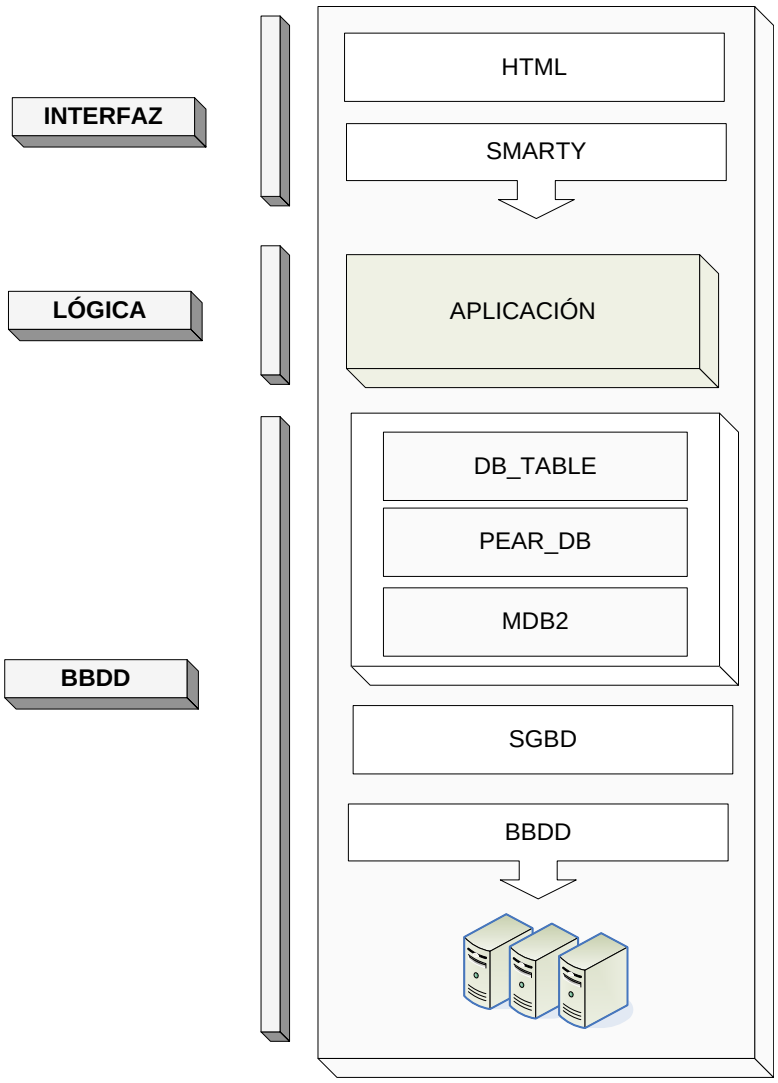


Figura 6.1: Visión general del sistema

En cuanto a las funcionalidades que ofrece el sistema se dividen en grupos semánticos:

Gestión de usuarios: Incluye el alta, baja, consulta y modificación de usuarios y grupos. También incluye la asignación de operaciones del sistema a los grupos.

Gestión de colegios: Se gestiona todas las operaciones relacionadas con los colegios y de las personas de contacto asociadas a ellos. También incluye la asignación de facultades a colegios [1].

Gestión de visitas: Incluye la parte de alta visita y consultar/buscar visita. Es posible consultar el colegio donde se va a realizar la misma, la persona de contacto, el conductor y los informes oportunos.

[1] No desarrollado en esta versión

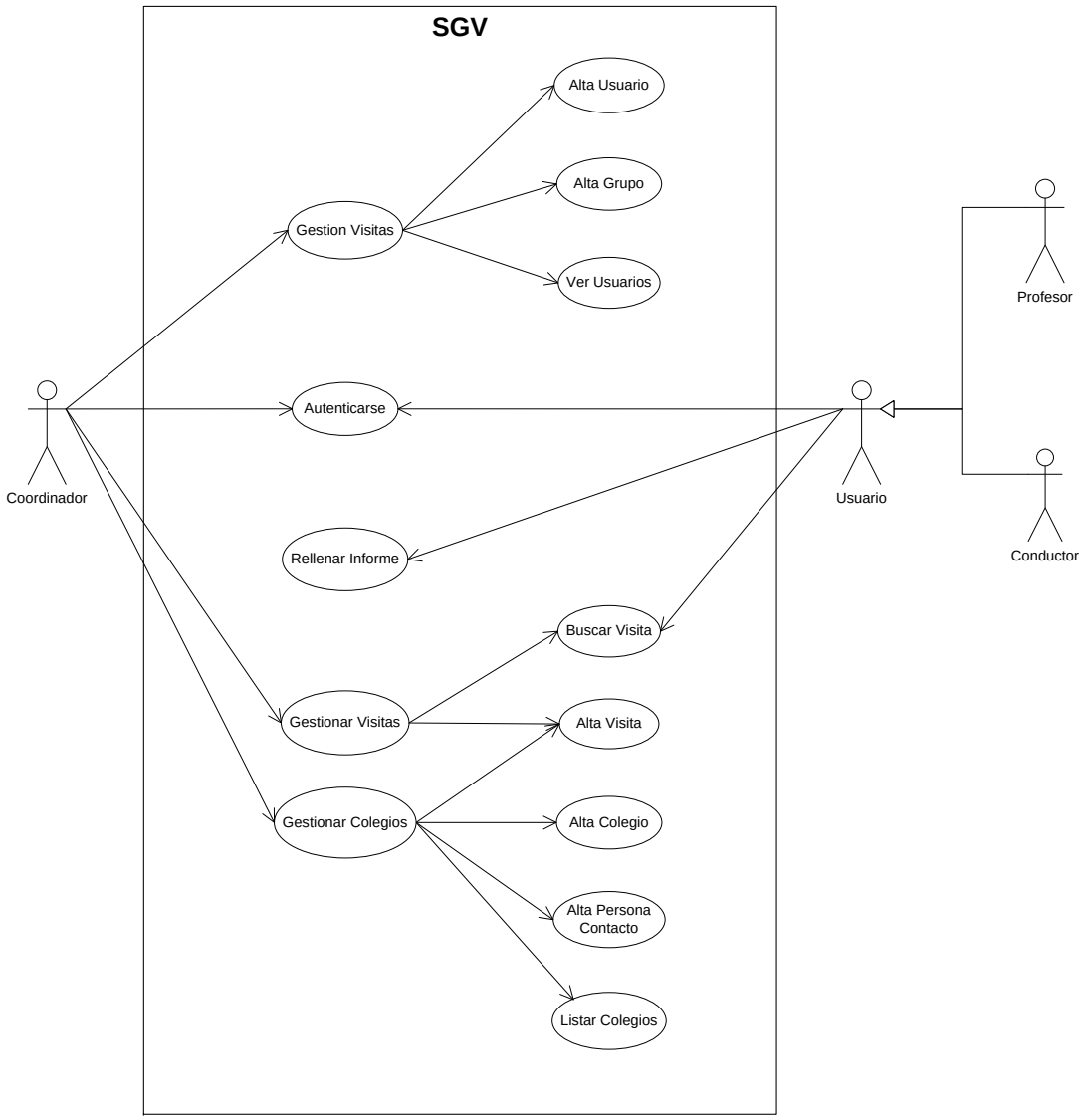


Figura 6.2: Casos de uso

6.1.2. Modelo Conceptual

Una parte de la investigación sobre el dominio del problema consiste en identificar los conceptos que lo conforman. Para representar estos conceptos se usa el diagrama de estructura estática de UML, que se denomina modelo conceptual. En este modelo se representan conceptos del mundo real, no de componentes software.

El objetivo de la creación de un modelo conceptual es mejorar la comprensión del problema. Por tanto, a la hora de incluir conceptos en el modelo, es mejor incluir el mayor número posible para cubrir el mayor espectro del conocimiento del problema.

La creación del modelo conceptual involucra los siguientes pasos:

Identificar conceptos. Para identificar conceptos hay que basarse en la especificación de requisitos y en el conocimiento general acerca del dominio del problema. Otro consejo para identificar conceptos consiste en buscar sustantivos en los documentos de requisitos o, más concretamente, en la descripción de los casos de uso.

Para nombrar los conceptos se puede usar los siguientes tres puntos:

- Usar los nombres existentes en el territorio: Hay que usar el vocabulario del dominio para nombrar conceptos y atributos.
- Excluir características irrelevantes: Al igual que el cartógrafo elimina características no relevantes según la finalidad del mapa (por ejemplo, datos de población en un mapa de carreteras), un modelo conceptual puede excluir conceptos en el dominio que no son pertinentes.
- No añadir cosas inexistentes. Si algo no pertenece al dominio del problema no se añade al modelo.

Añadir las asociaciones. Una asociación es una relación entre conceptos que indica una conexión con sentido y que es de interés. Se incluyen en el modelo las asociaciones siguientes:

- Asociaciones para las que el conocimiento de la relación necesita mantenerse por un cierto período de tiempo (asociaciones 'necesita-conocer').
- Asociaciones derivadas de la lista de asociaciones típicas que se muestra en la Tabla 6.1.

Una vez identificadas las asociaciones se representan en el modelo conceptual con la cardinalidad adecuada.

Añadir los atributos. Es necesario incorporar al modelo conceptual los atributos necesarios para satisfacer las necesidades de información. Los atributos deben tomar valor en tipos simples (número, texto, etc.), pues los tipos complejos deberían ser modelados como conceptos y ser relacionados mediante asociaciones.

Incluso cuando un valor es de un tipo simple es más conveniente representarlo como concepto en las siguientes ocasiones:

- Se compone de distintas secciones. Por ejemplo: un número de teléfono, el nombre de una persona, etc.
- Tiene operaciones asociadas, tales como validación. Ejemplo: NIF.
- Tiene otros atributos. Por ejemplo un precio de oferta puede tener fecha de fin.
- Es una cantidad con una unidad. Ejemplo: El precio, que puede estar en pesetas o en euros.

Representarlos en un diagrama.

Categoría	Ejemplos
A es una parte física de B	Ala - Avión
A es una parte lógica de B	ArtículoVenta - Venta
A está físicamente contenido en B	Artículo - Estantería Pasajero Avión
A está lógicamente contenido en B	DescripciónArtículo - Catálogo
A es una descripción de B	DescripciónArtículo - Artículo
A es un elemento en una transacción o un informe B	TrabajoReparación - RegistroReparaciones
A es registrado/archivado/capturado en B	Venta - TerminalCaja
A es un miembro de B	Cajero - Supermercado Piloto - Compañía_Aerea
A es una subunidad organizativa de B	Sección - Supermercado Mantenimiento - Compañía_Aerea
A usa o gestiona B	Cajero - TerminalCaja Piloto Avión
A comunica con B	Cliente - Cajero EmpleadoAgenciaViajes - Pasajero
A está relacionado con una transacción B	Cliente - Pago Pasajero - Billete
A es una transacción relacionada con otra transacción B	Pago - Venta Reserva - Cancelación
A está junto a B	Ciudad - Ciudad
A posee B	Supermercado - TerminalCaja Compañía_Aérea - Avión

Tabla 6.1: Lista asociaciones típicas

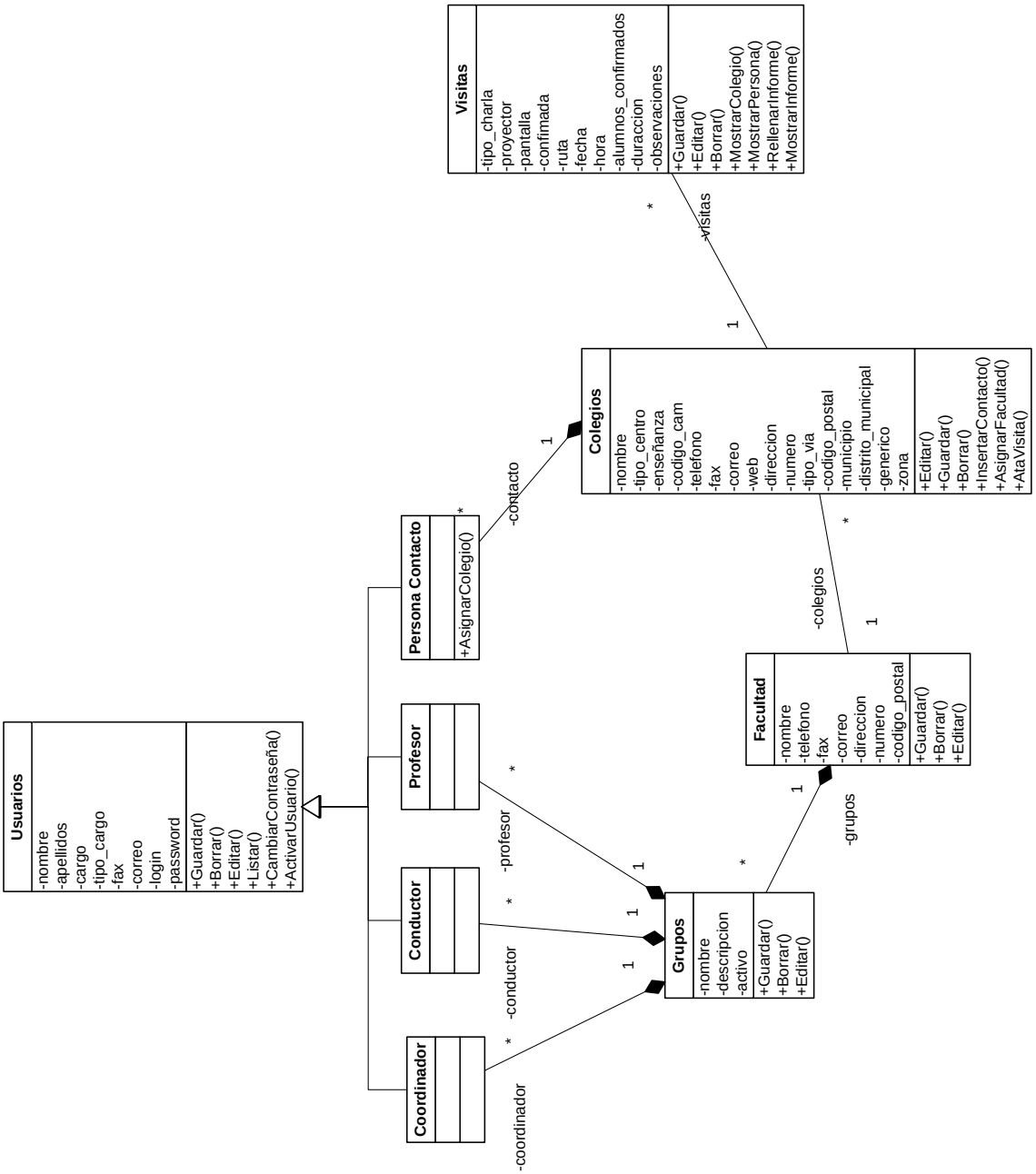


Figura 6.3: Diagrama del modelo conceptual

6.1.3. Modelo de Navegación del método OOWS

Como se comentó en la Subsección 4.2.4, para contruir el modelo de navegación se van a realizar dos tareas; la clasificación e identificación de los usuarios y la contrucción de los mapas navegacionales.

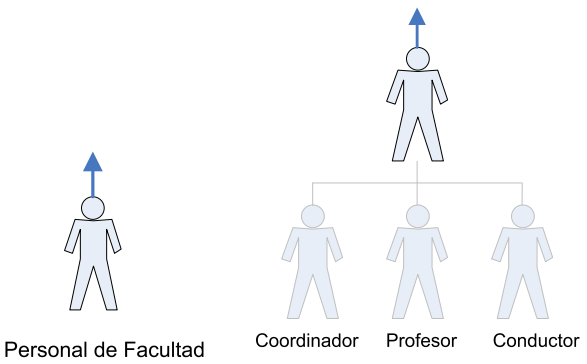
6.1.3.1. Clasificación e identificación de usuarios

En esta etapa se construye un diagrama de agentes (usuarios), donde se especifican qué tipos de usuarios van a poder interactuar con el sistema, qué interrelaciones existen entre ellos y cuál va a ser su modo de acceso al sistema.

En primer lugar se debe estudiar cuáles son los potenciales tipos de usuarios interactivos con el sistema y crear un agente por cada uno de ellos. Una vez identificados, se debe realizar un estudio para detectar interrelaciones entre ellos buscando propiedades comunes (acceso a funcionalidad común, visibilidad de la misma información, etc.) creando una taxonomía de usuarios, que permitiran reutilizar especificaciones navegacionales. Finalmente, cada tipo de agente puede ser de dos clases distintas:

- *Agentes instanciables.* Los agentes de este tipo constituirán los usuarios que la aplicación web reconocerá. Pueden ser de dos tipos: (1) aquellos que necesitan identificarse en el sistema (representados con el símbolo '↑' y (2) aquellos que pueden conectarse al sistema sin identificarse (símbolo '?'). Para este sistema todos los usuarios necesitarán conectarse con el sistema.

- *Agentes abstractos.* Este tipo de agente representa a agentes sin instancias directas. Es utilizado para llevar a cabo los mecanismos de especialización.



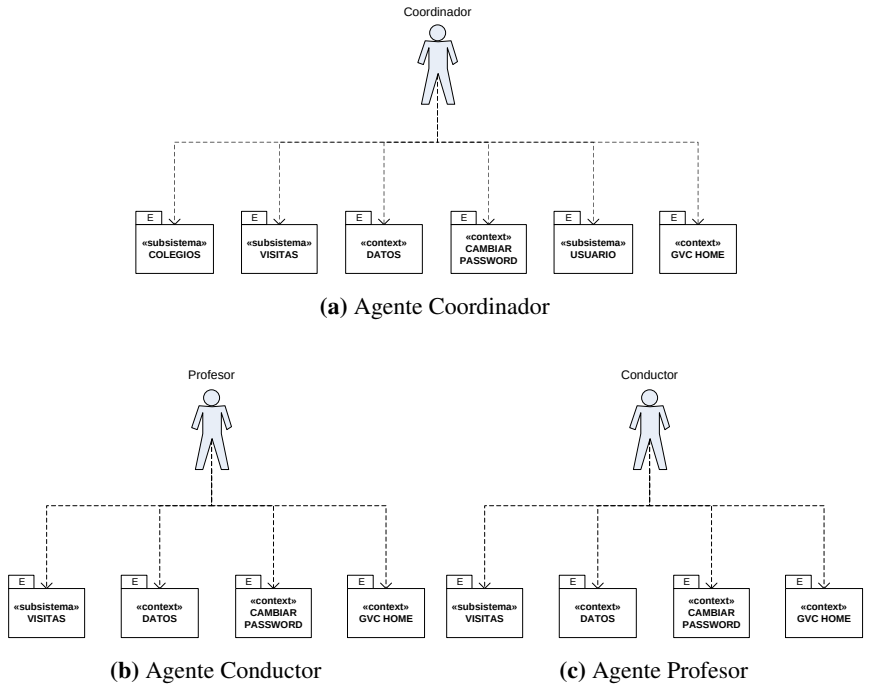
(a) Agente abstracto **(b)** Agente abstracto

Figura 6.4: Diagrama de usuarios de la aplicación web

En la Figura 6.4a se puede ver el agente *Personal de facultad* que tiene todos los privilegios de los internautas (de ahí su especialización), y en la Figura 6.4b, se muestran a los coordinadores, conductores y profesores. Todos estos usuarios precisan autenticación para entrar en el sistema y compartirán todos ellos los mismos privilegios como personal de facultad. Sin embargo, cada uno de ellos tendrá propiedades de navegación diferentes al definir su mapa de navegación propio.

6.1.3.2. Construcción de los Mapas Navegacionales

La siguiente fase consiste en construir los mapas navegacionales, uno por cada agente (usuario). Estos mapas se definen mediante un grafo dirigido en el que los nodos representan unidades de interacción con el usuario (gráficamente representado usando la misma notación que los paquete UML) y los arcos representan los caminos navegacionales válidos entre nodos (representados mediante flechas).



(a) Agente Coordinador

(b) Agente Conductor **(c)** Agente Profesor

Figura 6.5: Mapa navegacional de los agentes del sistema

La Figura 6.5a, muestra el mapa navegacional de los coordinadores. Este mapa define su visión del sistema que le proporciona acceso a un conjunto de nodos navegacionales sobre usuarios, colegios y visitas. De igual forma en las figuras Figura 6.5b y Figura 6.5cn se muestran los mapas navegaciones de los agentes conductor y profesor, respectivamente.

Cada uno de estos nodos, donde el usuario interactúa con el sistema, se corresponde con un contexto navegacional. Los contextos navegacionales permiten definir una vista sobre información y funcionalidad

definida en el diagrama de clases. Pueden ser de dos tipos:

- *Exploración (etiquetados con una* 'E'*)*. Representan nodos alcanzables desde cualquier otro nodo. Este tipo de contexto define unos enlaces de navegación implícitos (líneas discontinuas) que surgen del agente hasta el contexto. Uno de estos enlaces pueden marcarse como *default* o *home* (con la etiqueta *'H'*), indicando que este contexto será alcanzado directamente cuando el usuario se conecte al sistema (*contexto GVC home*).

- *Secuencia (etiquetados con una* `'S'`*)*. Aquellos que sólo pueden alcanzarse a través de un determinado camino de navegación.

En el diseño de aplicaciones web complejas surge la necesidad de poder organizar o estructurar los distintos contextos navegacionales que se hayan definido. Para ello se introduce una nueva primitiva, el subsistema de navegación. Los subsistemas de navegación (estereotipados con la palabra reservada «*subsystem*») permiten realizar una agrupación lógica de contextos navegacionales u otros subsistemas de navegación que están estrechamente relacionados entre sí y que comparten propiedades navegacionales (ver Figura 6.5a, subsistemas colegio, usuario, visita). De la misma manera que los contextos navegacionales, los subsistemas de navegación pueden ser de dos tipos, de exploración o de secuencia, según su alcanzabilidad.

Los enlaces navegacionales (arcos del grafo) representan la alcanzabilidad entre nodos navegacionales, indicando los caminos navegacionales válidos por el sistema para un determinado usuario. Para acceder al sisitema, el usuario deberá identificarse como coordinador, conductor o profesor.

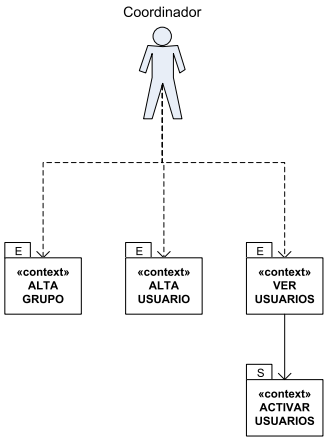


Figura 6.6: Estructura del subsistema `USUARIO`

En la Figura 6.6 se muestra el contenido del subsistema *usuario*. Este subsistema contiene tres contextos de exploración mediante los cuales se puede dar de alta a usuarios en el sistema, dar de alta a grupos y listar todos lo usuarios del sistema.

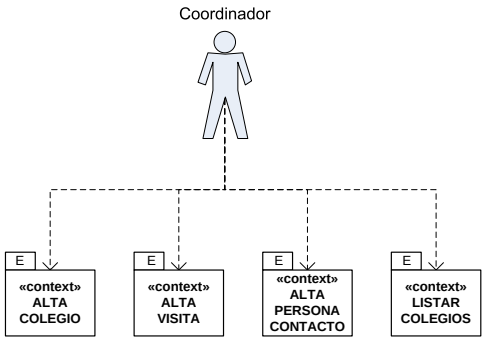


Figura 6.7: Estructura del subsistema `COLEGIO`

En la Figura 6.7 se muestra el contenido del subsistema *colegio*. Este subsistema contiene cuatro contextos de exploración mediante los cuales se puede dar de alta a un colegio, dar de alta una visita, dar de alta una persona de contacto y por último listar todos los colegios del sisitema.

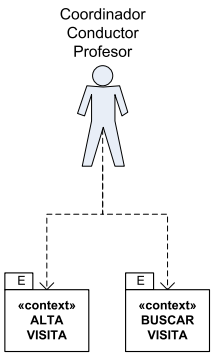


Figura 6.8: Estructura del subsistema `VISITA`

En la Figura 6.8 se muestra el contenido del subsistema *visita*. Este subsistema contiene dos contextos de exploración mediante los cuales se puede dar de alta una visita y realizar una busqueda de las visitas del sistema.

Una vez se ha definido la navegación de cada uno de los agentes detectados, mediante sus respectivos mapas navegacionales, el siguiente paso a realizar consiste en la definición de los contextos navegacionales.

En la Figura 6.9, la Figura 6.10 y la Figura 6.11 se observa la definición de los contextos para cada subsistema.

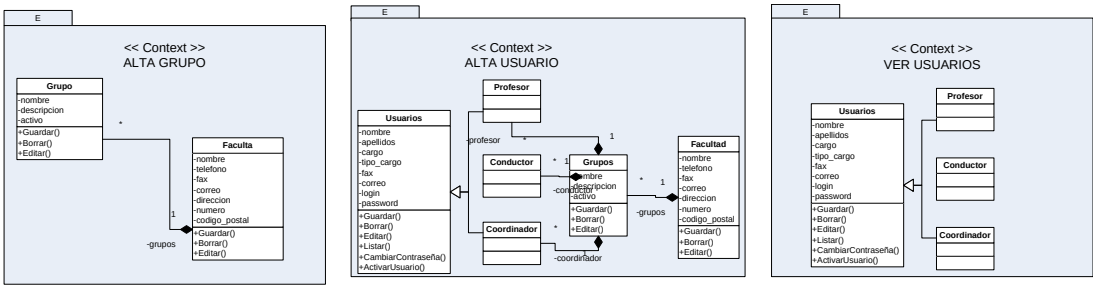


Figura 6.9: Definición de contexto del subsistema `USUARIO`

Un contexto navegacional está compuesto por un conjunto de clases navegacionales, que hacen referencia a clases identificadas en el diagrama de clases. Con ellas se puede definir la visibilidad (vista) ofertada al agente en este nodo, tanto de los atributos de la clase como de los servicios que puede activar. En un contexto navegacional debe aparecer al menos una clase navegacional principal, llamada clase directora *grupo* en el contexto `GRUPO` del subsistema `USUARIO` y opcionalmente otras que complementan la información de esta clase, llamadas clases complementarias (). Las clases navegacionales están unidas entre sí por relaciones binarias unidireccionales que pueden ser definidas sobre una relación existente entre las dos clases en el diagrama de clases.

Por ejemplo, en el subsistema `USUARIO`, se puede ver el contexto `ALTA USUARIO` que muestra información relativa a los usuarios del sistema así como de los grupos.

6.1.4. Modelo de Presentación

Como se comentó en la Subsección 4.2.5, una vez definido el modelo de navegación se especifican las características de presentación del sistema. Este modelo utiliza los nodos o contextos navegacionales como entidades básicas para definir las propiedades de presentación de información.

6.1.4.1. Implementación de la aplicación a partir del modelo Conceptual

Para realizar la implementación de aplicaciones web a partir del modelo conceptual realizado, se propone una arquitectura de tres capas. Este tipo de arquitectura permite estructurar claramente las aplicaciones web facilitando la adaptabilidad y extensibilidad de las mismas y ofrece una mayor escalabilidad. Es aconsejable utilizar este tipo de arquitectura en aplicaciones de gran tamaño y con un gran número de usuarios trabajando concurrentemente teniendo en consideración la escalabilidad frente a los requisitos de ejecución. La arquitectura propuesta se divide en las siguientes tres capas:

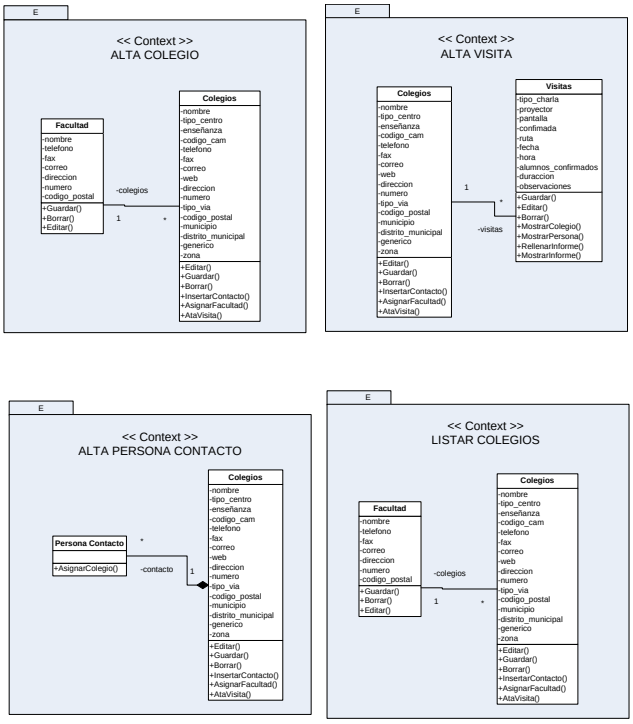


Figura 6.10: Definición de contexto del subsistema `COLEGIO`

Capa de presentación. Incluye la interfaz gráfica de usuario (en este caso, páginas web) para interactuar con el usuario (visualizando información, proporcionando acceso a servicios y facilitando la navegación).

Capa de aplicación. Esta capa implementa la lógica de negocio y la funcionalidad de consulta.

Capa de datos. Esta capa implementa el acceso a datos ocultando a las capas superiores detalles de los repositorios de información.

Partiendo del modelo de navegación, es posible obtener de forma sistemática un esqueleto de una aplicación web formado por un conjunto de páginas web interconectadas entre sí. Este conjunto de páginas web define la interfaz de usuario de la aplicación web para la navegación, la visualización y el acceso a su funcionalidad e información.

Añadiendo a esta información la especificación construida en el modelo de presentación podrá definirse un modo de visualización de esta información. Para construir este conjunto de páginas, se ha optado por utilizar HTML/CSS. El motivo principal de utilizar esta tecnología ha sido la flexibilidad que nos aportan las hojas de estilo CSS para cambiar características estéticas de las páginas web una vez construidas.

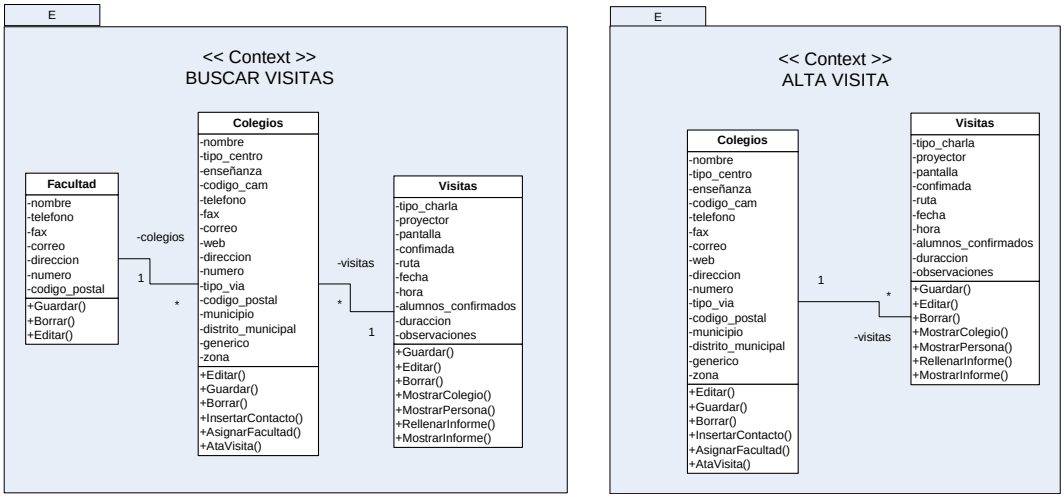


Figura 6.11: Definición de contexto del subsistema `VISITA`

Inicialmente se diseña la página de inicio o *login* mostrada en la figura Figura 6.12. Posteriormente por cada contexto navegacional o subsistema definido en el mapa de navegación se crea una página web.

Figura 6.12: Página de inicio de la aplicación

Los subsistemas únicamente modelan una agrupación lógica de contextos. Por tanto, las páginas generadas a partir de ellos contendrán únicamente la implementación de un menú a partir del cual poder acceder a cada uno de los contextos de exploración contenidos en el subsistema.

En las siguientes figuras, Figura 6.14 y Figura 6.15, se muestra las páginas que se implementan a partir de los contextos navegacionales contenidos en los subsistema definidos en la Subsubsección 6.1.3.2.

Figura 6.13: Presentación de la aplicación

6.1.5. Diseño de la interfaz de usuario

La interfaz de usuario es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.

El diseño de la interfaz es uno de los elementos clave en la realización del programa. Esta importancia toma un relieve especial en las aplicaciones web. La interfaz de usuario puede definirse como:

> Conjunto de trabajos y pasos que seguirá el usuario, durante todo el tiempo que se relacione con el programa , detallando lo que verá y escuchará en cada momento, y las acciones que realizará, así como las respuestas que el sistema le dará [Perea, 1996].

Sus principales funciones son:

- Manipulación de archivos y directorios
- Herramientas de desarrollo de aplicaciones
- Comunicación con otros sistemas
- Información de estado
- Configuración de la propia interfaz y entorno

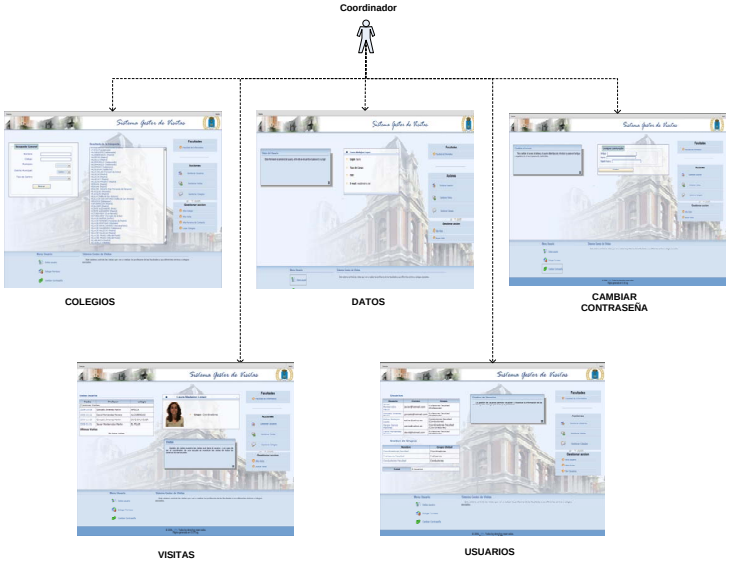


Figura 6.14: Contexto navegacional del coordinador

- Intercambio de datos entre aplicaciones
- Control de acceso
- Sistema de ayuda interactivo

En la Figura 6.16 en la que se muestra el diseño que se ha seguido para hacer la interfaz se diferencian tres partes; la cabecera, el cuerpo y el pie. La parte del cuerpo a su vez se divide en dos partes; por un lado esta la zona principal, donde se muestran la mayoría de los datos del sistema y por otro lado esta el el menú lateral, donde se encuentran las diferentes opciones con las que el usuario puede interaccionar con el sistema.

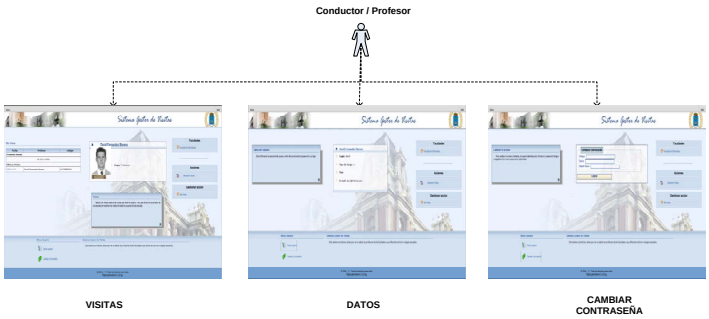


Figura 6.15: Contexto navegacional del conductor y profesor

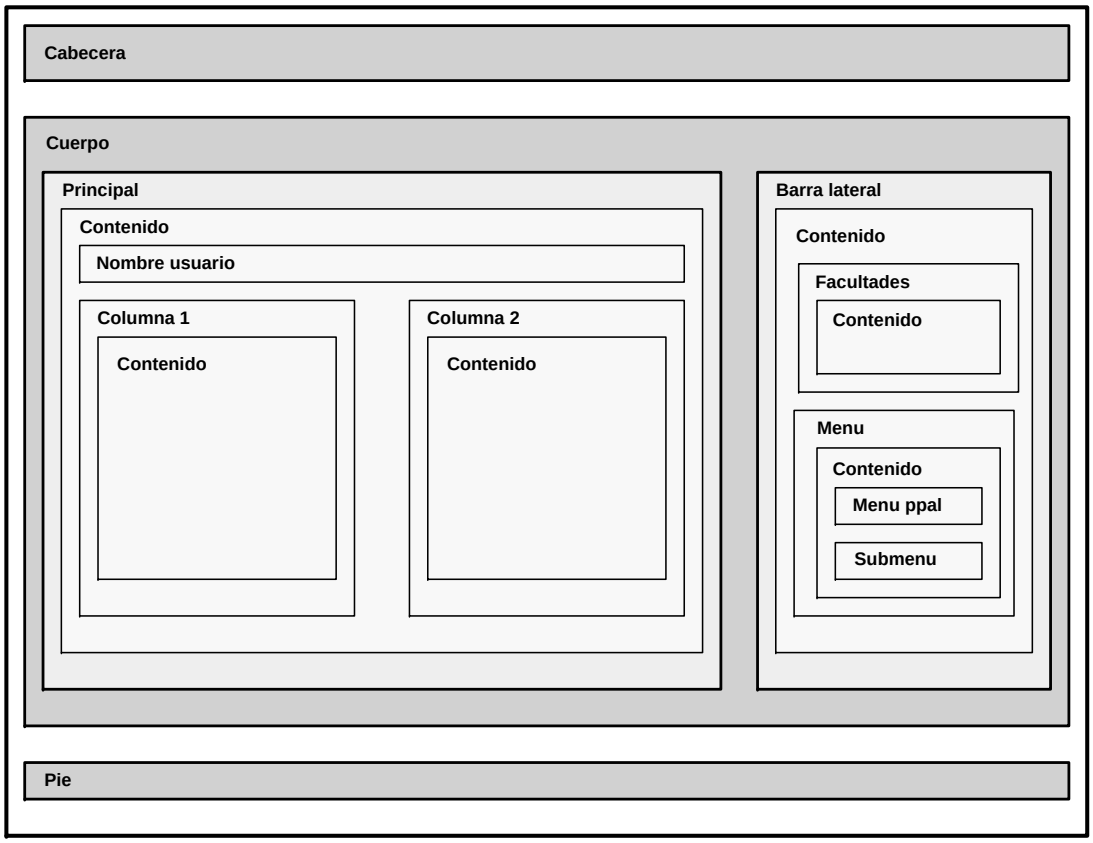


Figura 6.16: Estilo incial de la aplicacion Web

6.1.5.1. Diagrama de módulos

Un diagrama de módulos muestra la localización de objetos y clases en módulos del diseño físico de un sistema.Un diagrama de módulos representa parte o la totalidad de la arquitectura de módulos del sistema.

Como se puede ver en la Figura 6.17, el módulo principal de la aplicación es `gvc.class.php`, que se encargará de inicializar todo el sistema. El resto son operaciones del sistema o bibliotecas (PEAR, *Smarty*).

En el diseño físico se tiene por un lado los módulos `lib/*tablas.php`, que contienen la información de las tablas del sistema, el módulo `lib/session.php`, que contiene información de la sesión que se esta llevando a cabo en la aplicación, el módulo `script.php`, para la generación del código *JavaScript* y AJAX y el módulo `index.php` que se encarga de implementar el punto de entrada de la lógica de la aplicación.

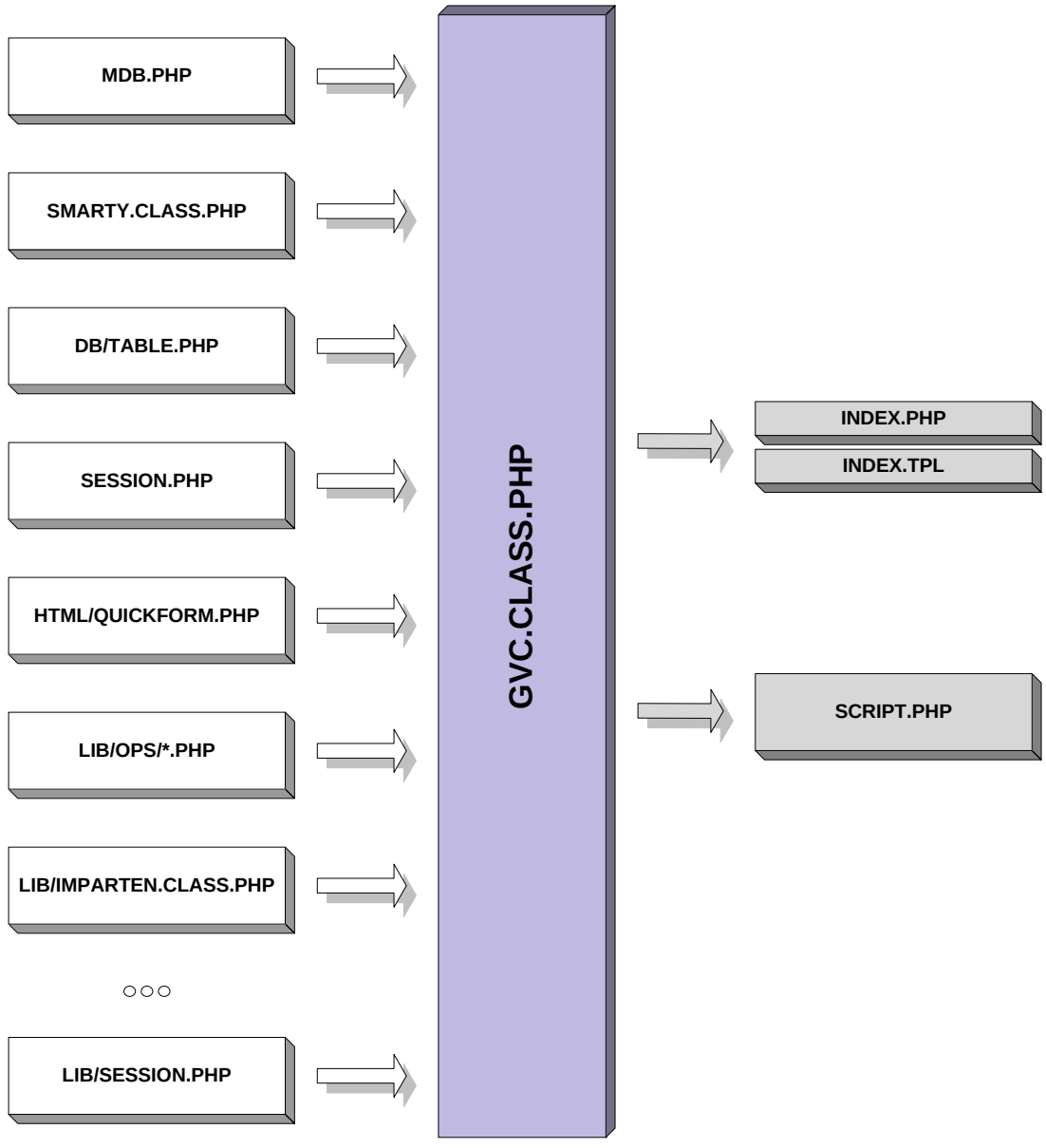


Figura 6.17: Diagrama de módulos

6.1.6. Jerarquía de ficheros

Otro punto interesante va a ser la estructura de ficheros y directorios que se puede ver en la Figura 6.18:

Carpeta `gvc`. Esta es la carpeta principal o raíz de la aplicación.

Carpeta `cfg`. Almacena información de configuración.

Carpeta `css`. Es la carpeta donde se encuentran los ficheros que definen el estilo de la aplicación.

Carpeta `images`. En esta carpeta se encuentran todas las imágenes que se han utilizado en la aplicación.

Fichero `login`. Este es el fichero de la pantalla inicial, donde el usuario deberá autenticarse para entrar en el sistema.

Fichero `index`. Punto de acceso cuando el usuario está autenticado.

Fichero `script`. Generación del código *JavaScript* / AJAX necesario.

Carpeta `lib` . Contiene los ficheros de bibliografía que implementan la lógica de aplicación y acceso a datos.

- *Carpeta* `lng`. Almacena definiciones del lenguaje para soporte multilenguaje.
- *Carpeta* `ops`. Es una carpeta donde se encuentran todas las operaciones que se pueden hacer en el sistema.
- *Carpeta* `tpl`. En esta carpeta están los esqueletos de presentación de cada una de las operaciones. Por ejemplo, tendremos un fichero *UsuarioDetalles.tpl* que va a definir la estructura o forma que tendrá el fichero *UsuarioDetalles.php*.
- *Carpeta* `smarty`. Esta carpeta contiene la información de configuración para poder utilizar las plantillas *Smarty*.
- *Carpeta* `Pear`. Al igual que la carpeta anterior, esta carpeta contendrá todos los módulos PEAR necesarios en nuestra aplicación.

6.2. Diseño de datos

En este apartado, se analizará tanto el diagrama entidad / relación como su paso a tablas.

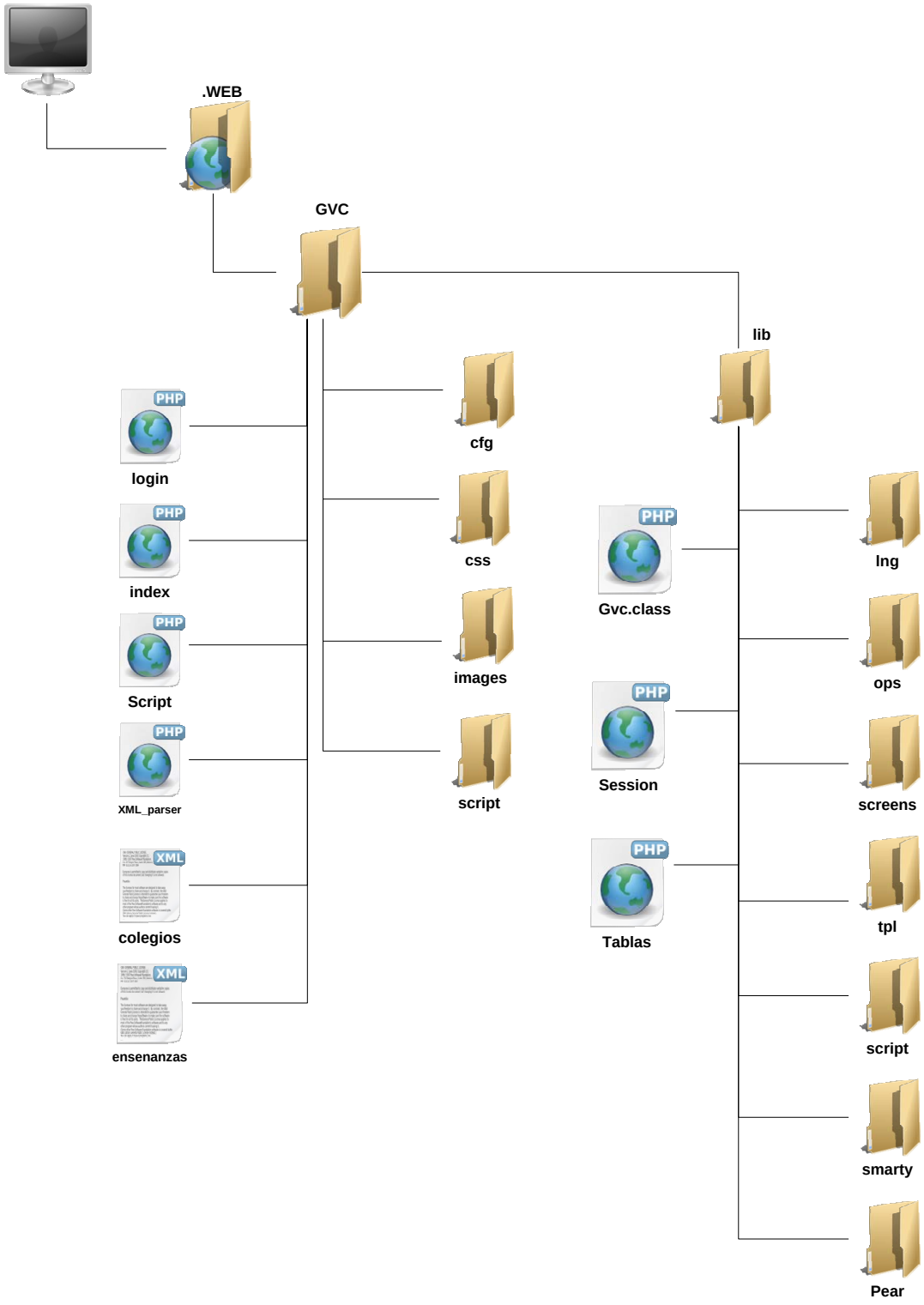


Figura 6.18: Jerarquía de ficheros

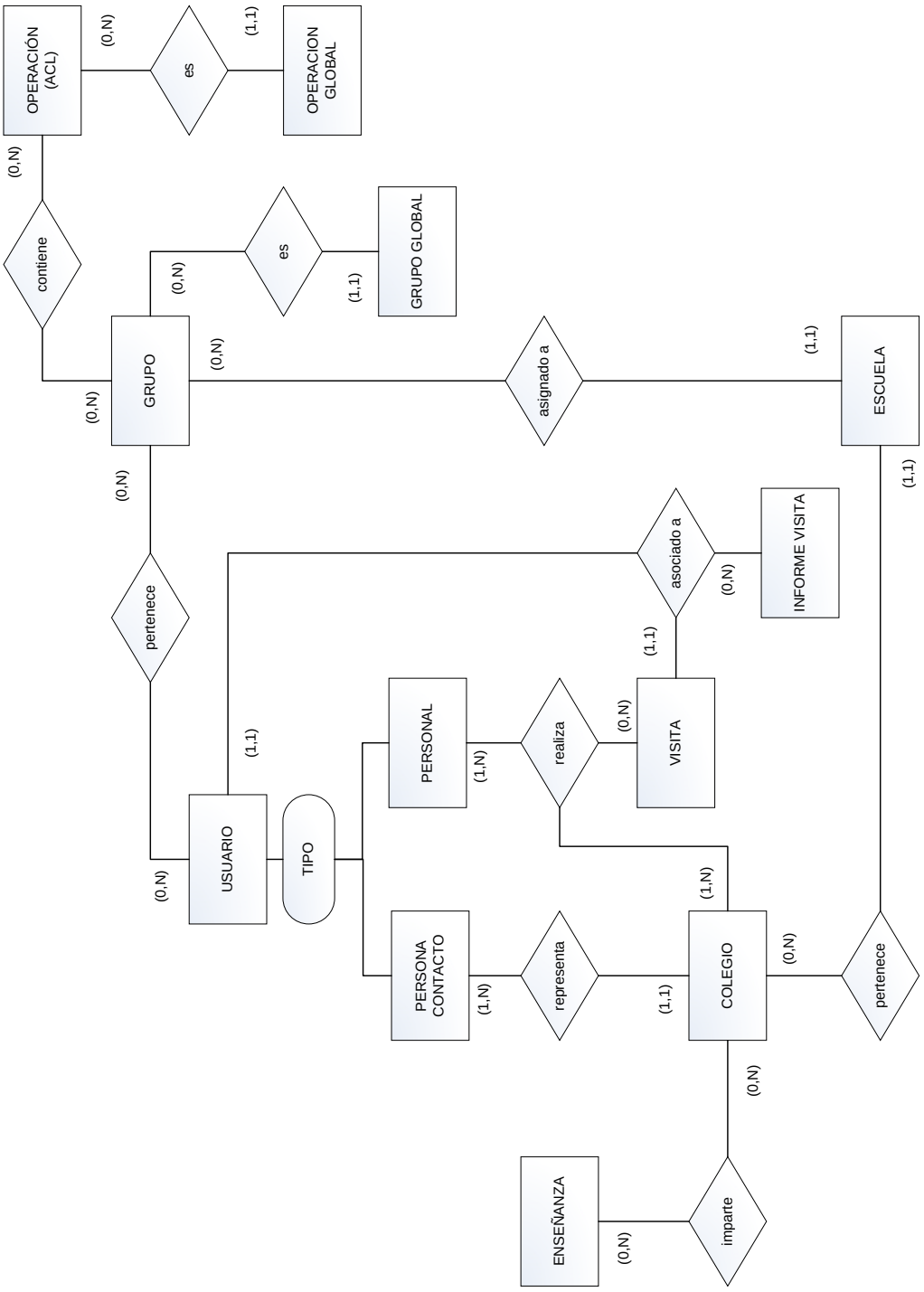
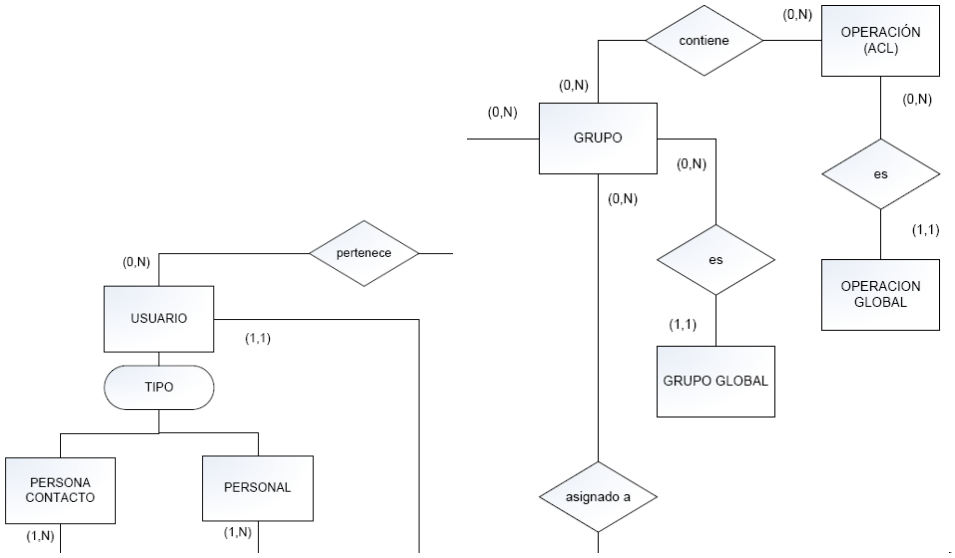
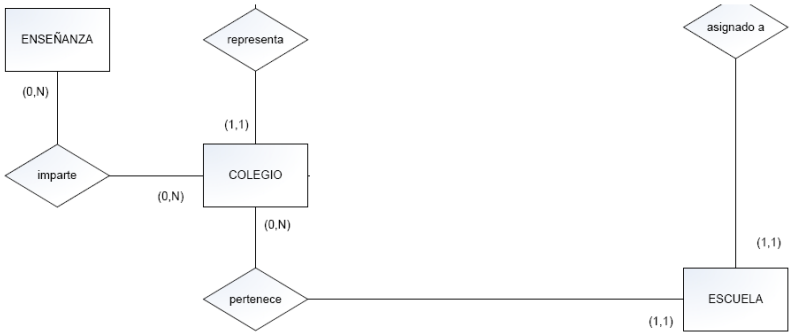


Figura 6.19: Modelo entidad / relación

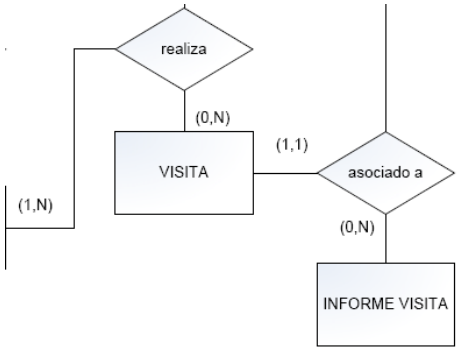


(a) Parte `USUARIOS`

(b) Parte `GRUPOS`



(c) Parte `COLEGIOS`



(d) Parte `VISITAS`

Figura 6.20: Partes del modelo E-R

6.2.1. Modelo entidad / relación

En este caso, los datos se han modelado según el diagrama E/R mostrado en la Figura 6.19. Este diagrma se compone de cuatro bloques básicos:

- Gestión de los usuarios del sistema, la cual esta formada por la jerarquía `USUARIO` (Figura 6.20a).
- Gestión de los grupos que se crean en el sistema, junto con las operaciones que se asignan a cada grupo (Figura 6.20b).
- Gestión de colegios, la cual engloba las enseñanzas de cada colegio y la pertenencia con las escuelas o facultades (Figura 6.20c).
- Gestión de las visitas (Figura 6.20d).

6.2.2. Paso a tablas

Al igual que en el diseño E-R de la figura Figura 6.19, en el paso a tablas (Figura 6.21) se mantiene los cuatro bloques principales.

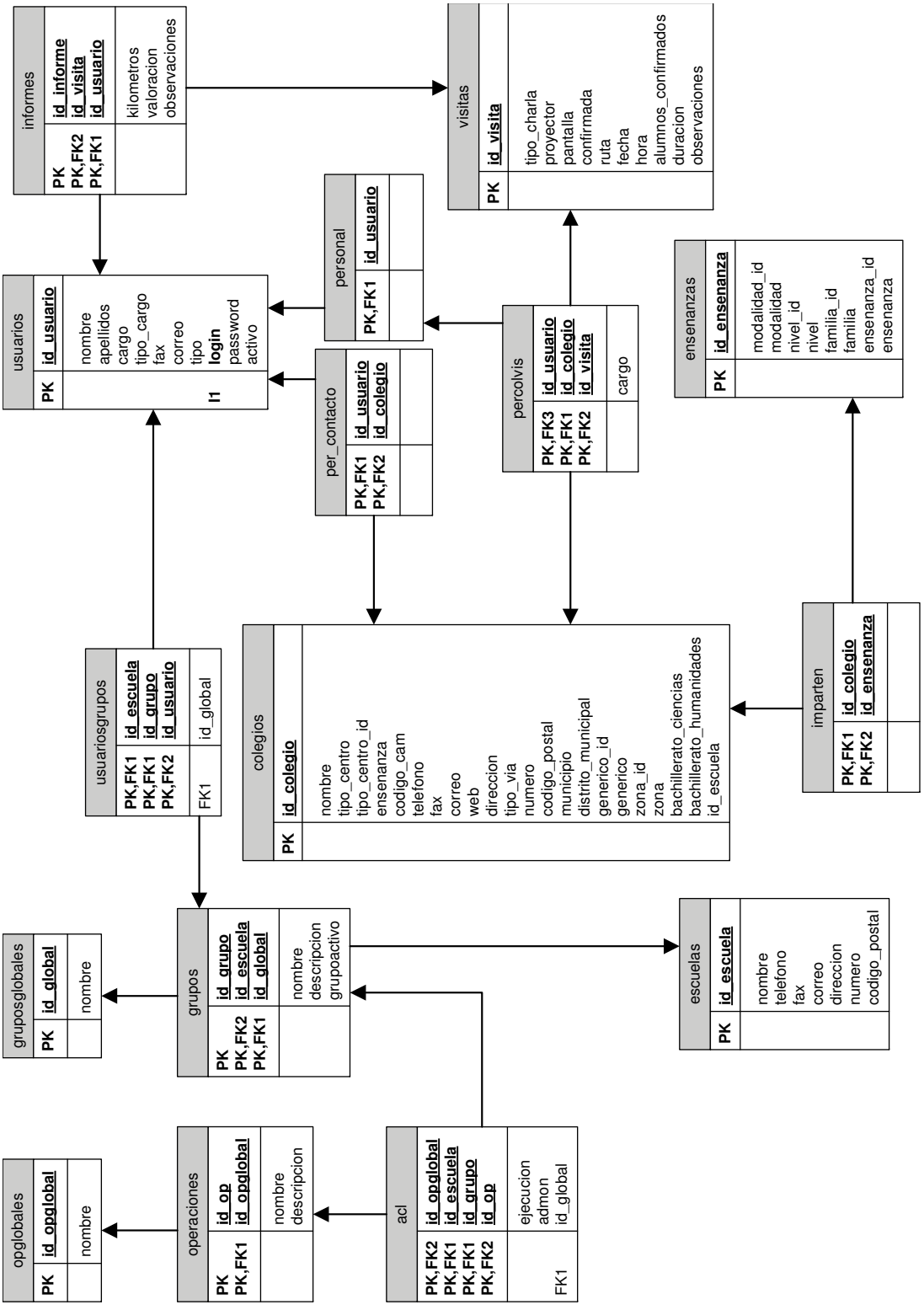


Figura 6.21: Paso a tablas

Capítulo 7

Desarrollo de la aplicación

Tras la planificación, formalización de los requisitos y diseño, es necesario plasmar la solución en un lenguaje de programación.

De forma análoga al diseño, en el que se abstraen patrones para aplicarlos a otros proyectos, durante la implementación se pueden abstraer detalles que pueden simplificar desarrollos similares. Estos detalles de implementación son recopilados para su uso en futuros proyectos.

Además, durante el desarrollo se han generado una serie de ficheros de código, documentación que han precisado de cierto tiempo para completarse. Un análisis del tiempo invertido en cada una de las fases, junto al tamaño del programa final desarrollado, permitirá mejorar la precisión de la planificación y asignación de recursos en proyectos similares futuros.

7.1. Implementación

La implementación de esta aplicación web se ha realizado en PHP generando código HTML, CSS y *Javascript* que es interpretado por el navegador de usuario. Durante el desarrollo han surgido algunas dificultades cuyas soluciones pueden ser de interés en el futuro.

7.1.1. Generación y proceso de formularios en PHP

El poder de PHP reside en su simplicidad y velocidad. Una de las aplicaciones más comunes que se utilizan en este lenguaje son los formularios PHP, por su parte PHP no ofrece ninguna función para el desarrollo de los formularios.

HTML_QuickForm [PEAR, 2004c], es un modulo PEAR que simplifica el trabajo con formularios HTML de manera mas sencilla. En vez de mostrar los elementos uno por uno, el método define un contenedor de un formulario al que se añaden los distintos controles con una sintaxis muy sencilla.

A partir de este contenedor es posible mostrar el formulario y validar la respuesta del usuario, mostrar los mensajes de error, generar código *JavaScript* para validar en el cliente, simplifica la subida de ficheros en el formulario...

Este módulo PEAR se integra con otros módulos (por ejemplo, el módulo usado para la gestión de la base de datos, o las plantillas *Smarty* usadas en la presentación) para generar formularios de forma extremadamente sencilla y con una cantidad de código mínima. En la Figura 7.1 se muestra el formulario para el registro de usuarios en el sistema y su código PHP, utilizando HTML_QuickForm, correspondiente (Programa 7.1).

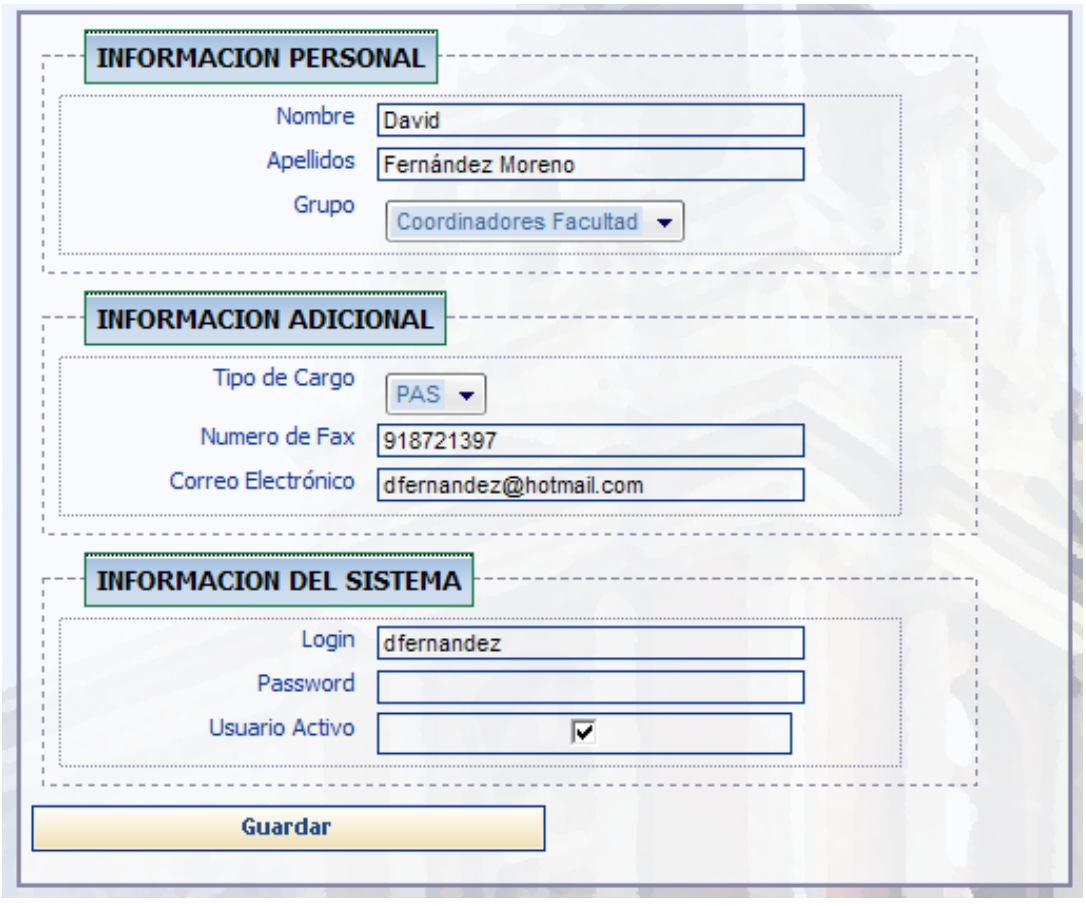


Figura 7.1: Formulario de registro de usuarios

Programa 7.1: Código del formulario de registro

```
1 <?php
2 require 'HTML/QuickForm.php';
```

```
3  $form = new HTML_QuickForm();
4
5  $renderer =& new HTML_QuickForm_Renderer_ArraySmarty($gvc->tpl);
6
7  // build the HTML for the form
8  $form =& $gvc->Usuarios->getForm();
9  $form->addElement('select', 'grupos','Grupo');
10 $form->addElement('reset','borrar','Borrar');
11 $form->accept($renderer);
12
13 // assign array with form data
14 $gvc->tpl->assign('form_data', $renderer->toArray());
15
16 ?>
```

Cuando el formulario es enviado se valida el campo usuario, en caso de que el formulario sea enviado sin haber escrito un valor en la caja de texto, se mostrará el formulario con un mensaje de que ese campo es requerido. Cuando se envía el formulario con datos en la caja de texto, se procesan los datos en la función validar( ) para posteriormente mostrar los datos introducidos como una salida del programa de manera satisfactoria.

7.1.2. Tabla ACL

La ACL es una tabla en la que se recopilan los derechos de acceso que cada usuario posee para un objeto determinado, como directorios, ficheros, puertos, etc. En este caso, se ha utilizado para tener un control sobre las operaciones que pueden realizar los distintos usuarios según el grupo al que pertenezcan.

En la Figura 7.2, se diferencian dos partes, en la parte de arriba, se tiene información del grupo, como es el nombre, la escuela a la que pertenece, el grupo global y una descripción. En la parte de abajo esta la tabla ACL, donde por un lado están todas las operaciones del sistema y por cada una de ellas hay un *tick* que indica si sobre esa operación el grupo tiene permisos de ejecución y/o administración.

7.1.3. Autenticación de usuarios

El mecanismo básico de seguridad es la autenticación del usuario a partir de la cual se le asignan los privilegios. La autenticación está basada en la asociación de un *login* (usuario) y una palabra clave, por lo que la protección de esta palabra es vital.

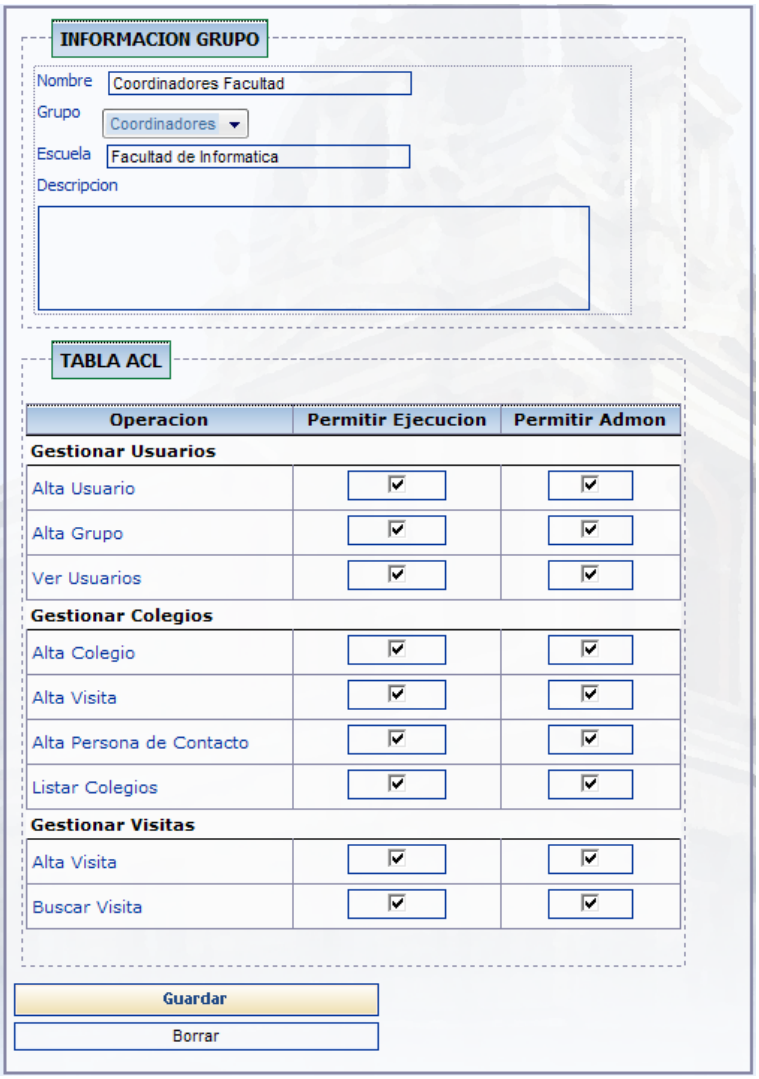


Figura 7.2: Edición de la tabla ACL para un grupo.

Para proteger la contraseña de usuario este se almacena utilizando un algoritmo de *hash*. Una función resumen (*hash*) tiene las siguientes características:

- Todos los números resumen generados con un mismo método tienen el mismo tamaño sea cual sea el texto utilizado como base.
- Dado un texto base, es fácil y rápido calcular su numero resumen
- Es imposible reconstruir el texto base a partir del numero resumen
- Es casi imposible que dos textos base diferentes tengan el mismo numero resumen

Hay muchos algoritmos de este tipo. Los más conocidos son MD5 y SHA, que se utilizan habitualmente para firmas digitales.

Programa 7.2: Código para autenticarse en el sistema

```
1
2  function Login($usuario,$password) {
3          global $gvc, $LNG;
4
5          // Comprobamos que los parametros del usuario son corrector
6          $password = md5($password);
7
8          $filter= "login = '$usuario' and password='$password' and activo='1'";
9          $user=$gvc->Usuarios->select('todos_usuarios',$filter);
10         if ( count($user)==0 ) {
11             $gvc->Error('error', 'ERROR_LOGIN');
12                         return false;
13         }
14         $_SESSION['usuario']=$user[0];
15
16         // Buscar los grupos
17         $id = $_SESSION['usuario']['id_usuario'];
18         $filter = "usuarios.id_usuario='$id'";
19         $grupos=$gvc->Usuarios->select('grupos_usuario',$filter);
20         $_SESSION['usuario']['grupos']=$grupos;
21
22         // Buscar las facultades
23         $id = $_SESSION['usuario']['id_usuario'];
24         $filter = "usuarios.id_usuario='$id'";
25         $facultades=$gvc->Usuarios->select('facultades_usuario',$filter);
26         $_SESSION['usuario']['facultades']=$facultades;
27
28         return true;
29 }
30
31
32 function Actualizar($data,$new,$personal) {
33         global $gvc;
34         $gvc->Usuarios->db->beginTransaction();
35
36         // Añadimos Nuevo Usuario
37         $usuario=$data['login'];
38         $filter= "login = '$usuario' and activo='1'";
39         $user=$gvc->Usuarios->select('todos_usuarios',$filter);
```

```
40          if ( count($user)>0 ) {
41                  return $result=3;
42          } // if
43          $nombre=$data['nombre'];
44          $id=$gvc->Usuarios->Select('dame_ultimo_id');
45          $id=$id+1;
46
47          $data['password'] = md5($data['password']);
48          $cols_vals = array(
49                  'id_usuario'=> $id,
50                  'nombre'    => $data['nombre'],
51                  'apellidos' => $data['apellidos'],
52                  'tipo_cargo'=> $data['tipo_cargo'],
53                  'fax'       => $data['fax'],
54                  'correo'    => $data['correo'],
55                  'login'     => $data['login'],
56                  'password'  => $data['password'],
57                  'activo'    => $data['activo'],
58          );
59
60          // Lo añadimos en la tabla usuarios
61          $result=$gvc->Usuarios->Insert($cols_vals);
```

Utilizar funciones de resumen en PHP es tan sencillo como invocar a la función de biblioteca *md5(string cad)*, que calcula el *hash* MD5 de una cadena. Para validar un usuario, el módulo de autenticación recibe la contraseña introducido por el usuario, a continuación, se le aplica la función *md5* y se comprueba si existe en la base de datos algún usuario con ese *login* y ese md5 almacenado en el atributo del *password*.

Con esto se evita que cualquier persona que tenga acceso a la base de datos pueda obtener la contraseña de los usuarios. Además, es útil para que en el caso de que la comunicación entre servidor web y servidor de bases de datos este siendo espiada, el atacante no obtenga la contraseña del atacado de forma clara, sino que lo que capturará será una clave cifrada que no servirá para entrar al sistema.

7.2. Análisis de coste y tiempo

Para la planificación o asignación de recursos de futuros proyectos es necesario realizar un análisis que permita determinar el tiempo que se precisa para realizar el proyecto en función de su tamaño. Además, es posible valorar el coste de realización del trabajo de cara a su posible comercialización.

En total hay 11.456 líneas de código y su desarrollo se ha prolongado por espacio de unos 5 meses, aproximadamente (de junio a octubre de 2006).

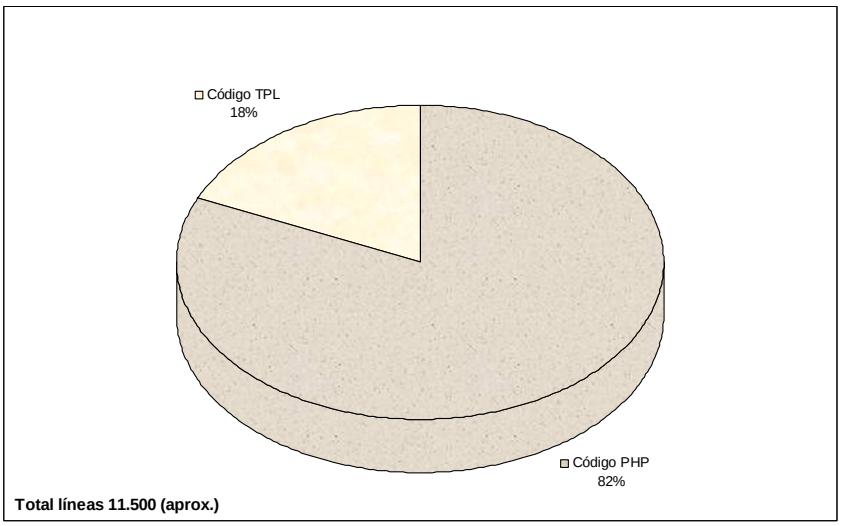


Figura 7.3: Líneas PHP / *Smarty*

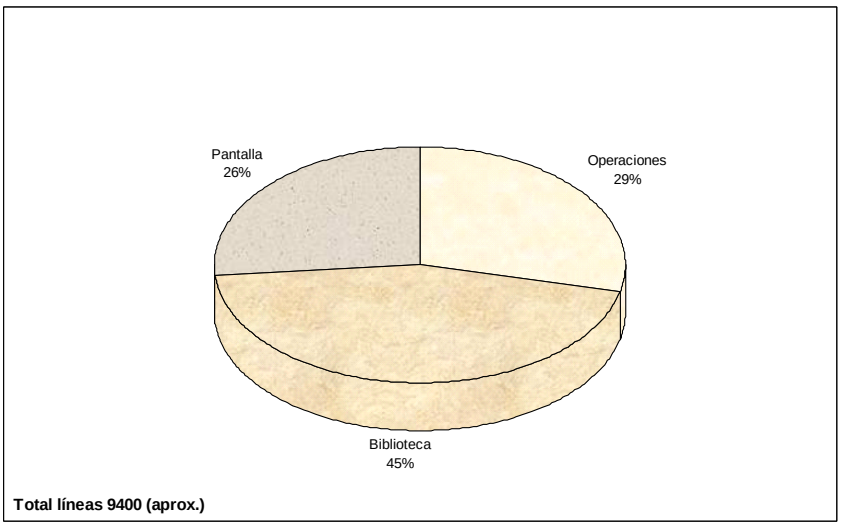


Figura 7.4: Partes del código PHP

La Figura 7.3, muestra una comparativa entre las líneas de código PHP, que implementan la lógica de la aplicación, y las líneas de los ficheros de *Smarty*, que definen el diseño de la misma. En la figura podemos ver como el número de líneas de código PHP es bastante mayor que el de las plantillas *Smarty*, lo que indica que el uso de *smarty* permite reducir al mínimo el esfuerzo dedicado a la interfaz de usuario con un resultado equivalente. Sin embargo, falta destacar el proceso de diseño de interfaz, por lo que el tiempo final dedicado a la presentación es algo superior.

Dentro del código PHP, la Figura 7.4 muestra las líneas dedicadas a la generación de interfaz (pantalla); gestión de tablas de la BBDD, ficheros de configuración y puntos de entrada (biblioteca) y, por último, las operaciones.

Según la figura, las líneas de código de la parte de biblioteca es en proporción el doble respecto a las líneas de pantalla y de operaciones.

El *COnstructive COst MOdel* (COCOMO) es un modelo que permite realizar estimaciones y planificaciones de proyectos de sistemas de información. Si aplicáramos el modelo COCOMO al trabajo realizado harían falta 32 personas-mes, con un coste total de 210.000 euros.

Parte IV

Conclusiones

Capítulo 8

Conclusiones y líneas futuras

Tras finalizar este trabajo, se dispone de una aplicación vía web que permite gestionar las visitas a colegios de una única facultad. Junto a esta aplicación, se dispone de una documentación y diseño de la base de datos y del código desarrollado.

Al finalizar todo el proceso es necesario realizar un análisis del desarrollo para lograr su máximo aprovechamiento. Fruto de este análisis se sintetizan los puntos clave, las dificultades, conocimientos adquiridos y la valoración de la experiencia.

Al comienzo de este trabajo se plantearon una serie de objetivos, como por ejemplo, que el sistema fuera fácilmente ampliable, de los cuales, la gran mayoría se han satisfecho, dejando algunas líneas abiertas para futuras implementaciones.

8.1. Conclusiones personales

Durante estos últimos 5 meses, que ha sido mas o menos lo que me ha llevado de tiempo este proyecto, sólo veo cosas positivas. En primer lugar decir, que me ha resultado una experiencia muy gratificante de la que he aprendido muchísimo. Muchas cosas que desconocía por completo y que después de todo este tiempo han despertado en mi mucha curiosidad, como son las tecnologías web y su problemática.

Hoy en día, las aplicaciones web tiene una importancia especial y es por ello por lo que pienso que me puede ayudar mucho durante mi vida profesional. Sin más, decir que han sido unos meses bastantes aprovechados y de los que estoy muy contento por todo los conocimientos que he adquirido.

8.1.1. Dificultades en el desarrollo

Alguna de las dificultades encontradas en el desarrollo de la aplicación han sido:

Elección del lenguaje de programación. Debido a mi desconocimiento de los numerosos lenguajes existentes para programar aplicaciones web, esta a sido una de las principales dificultades encontradas. Al final, el lenguaje utilizado para programar la aplicación fue PHP. Los motivos principales de esta elección fueron que ya había programado otras aplicaciones anteriormente y que PHP de daba soporte para desarrollar este proyecto.

Problemas con las base de datos. Otro punto del desarrollo que también planteo una dificultad, fue el echo de tener que desarrollar la aplicación independiente de la base de datos donde estuviera. La solución optada fue el utilizar unas librerías de PEAR, y concretamente el módulo llamado DB_TABLE, que abstrae la base de datos que haya instalada en la máquina donde se ejecute la aplicación.

Información de los centros educativos. Desde el inicio era necesario obtener la información de los centros educativos de la comunidad. Tras una búsqueda inicial se localizó el sitio web donde, la propia comunidad, proporciona esa información. Tras plantear diversas formas de extraer esa información se optó por descargar el CD de información y analizar los ficheros XML que contenían los datos de la aplicación web.

El resultado final es una pequeña utilidad, escrita en PHP, que es capaz de analizar el formato XML usado en ese CD y almacenarlos en la BBDD de la aplicación.

La solución optada para pasar toda esta información a la base de datos fue mediante un *parser*, es decir, un fichero el cual leyera todos esos archivos XML, verificara que su formato fuera correcto y mediante otro módulo PEAR, fuera pasando la información a las tablas de la base de datos.

8.1.2. Problemas de depuración

Los programas informáticos son a menudo imperfectos y, por lo tanto, el código puede contener diversos tipos de errores. La única forma de detectar los errores lógicos es probar el programa, ya sea manual o automáticamente, y comprobar que el resultado es el esperado. Las pruebas deben ser una parte integrante del proceso de desarrollo del software.

Desgraciadamente, aunque las pruebas pueden indicar que el resultado de un programa es incorrecto, normalmente no proporcionarán ninguna pista acerca de que parte del código ha causado realmente el problema. Es aquí donde cobra sentido el proceso de depuración.

Normalmente, aunque no siempre, la depuración implicará el uso de un depurador, una eficaz herramienta que permite observar el comportamiento del programa en tiempo de ejecución y determinar la ubicación de los errores semánticos. También es posible utilizar ciertas funciones de depuración integradas en el lenguaje y sus bibliotecas asociadas. Muchos programadores tienen su primer contacto con

la depuración cuando intentan aislar un problema agregando al código llamadas a funciones de salida, como *printf*. Esta técnica de depuración es perfectamente válida, pero hace que una vez encontrado y resuelto el problema sea necesario revisar el código para eliminar las llamadas a funciones adicionales. Ocasionalmente, puede encontrarse una situación en la que, al agregar código nuevo (incluso una simple llamada a *printf*), el comportamiento del código que se intenta depurar cambie.

El desarrollo de aplicaciones web tradicionalmente ha sido complejo de depurar ya que lenguajes como PHP tienen un soporte limitado para esta tarea. Es por esto que cobra especial importancia el diseñar un mecanismo que permita realizar esta ardua pero importante labor.

8.2. Conocimientos

Evidentemente en una carrera como Ingeniería Informática, no es posible infundir todas las enseñanzas que podría necesitar el alumno en su carrera profesional principalmente por falta de tiempo y de cambio continuo en la tecnología.

Hoy en día, el tema de las aplicaciones web, tecnológias web, aplicaciones cliente / servidor, etc.. están siendo muy usadas por mucha gente a la hora de realizar programas.

Durante la realización de este proyecto me encontrado con numerosos problemas, muchos de los cuales nunca los había estudiado durante los años de docencia. Por ejemplo, el cómo diseñar una aplicación vía web para facilitar el uso de la misma, o el conocimiento de lenguajes que se utilizan para desarrollar una aplicación web, como pueden ser, PHP, *Active Server Pages* (ASP), *JavaScript* o HTML.

Actualmente, las aplicaciones y tecnologías web son un área en auge debido al impulso que están sufriendo desde el boom de la web 2.0, con la aparición de servicios sociales *on-line*. Sin embargo, existe un vacío de asignaturas durante la carrera que traten el desarrollo y problemática de este tipo de aplicaciones.

8.3. Líneas Futuras

Al finalizar este trabajo quedan abiertas varias líneas de desarrollo entre las que se pueden destacar las siguientes:

- Como se comentó en la Capítulo 5, se deberá implementar la gestión de asignar facultades a colegios o viceversa.
- Tener un acceso a sistemas de mapas y/o rutas como *Google Maps* o *Maporama*, para situar los centros educativos de la comunidad.

- Generar estadísticas y gráficas sobre visitas o sobre profesores que dan visita.
- Envío de recordatorios de visitas de forma automática mediante correos, SMS... antes de la visita y al día siguiente a los participantes de la misma.
- Gestión de Pagos.
- Incluir la posibilidad de que las personas de contacto de los centros puedan acceder al sistema para consultar las visitas programas, solicitar visitas de ciertos centros...
- Verificar su funcionamiento sobre más gestores de BBDD.
- Gestionar el material que es necesario en cada visita (si se dispone de proyector, ordenador...)
- Gestionar el número de asistentes a cada visita
- Generar informes de las visitas previas al centro para planificar mejor una nueva visita.

Parte V

Anexos

Anexo A

¿Cómo añadir nueva funcionalidad?

Un objetivo fundamental era hacer que el sistema fuera fácilmente ampliable, es decir, que se pudieran añadir fácilmente nuevas funcionalidades. El sistema diferencia dos tipos de funcionalidades:

- Operación global: En el sistema actual se encuentra gestión de usuarios, colegios, y visitas.
- Acción: se trata de una operación a la que se accede una vez seleccionada la acción global.

Los pasos para incluir cada una de las funcionalidades difieren. A continuación, se pasará a describir los pasos necesarios para añadir una nueva operación global en el sistema.

1. *Insertar en la base de datos*. El primer paso a seguir, es añadir en la base de datos esa nueva operación, por ejemplo, si la operación se llamara 'Gestión de facultades', se debería hacer una inserción en la table *Operaciones Globales*, para añadirla.

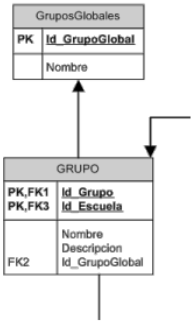


Figura A.1: Insertar la operación en la base de datos

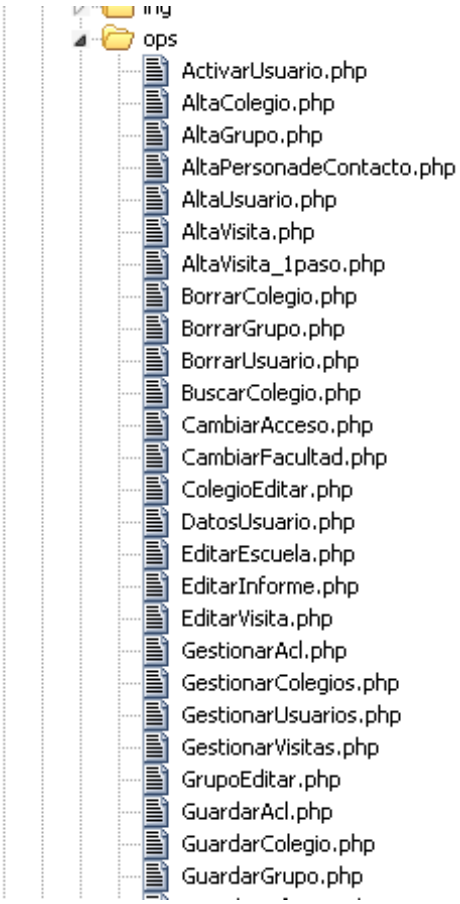


Figura A.2: Crear fichero de implementación

2. *Crear fichero de implementación*. Se crea un nuevo fichero dentro de la carpeta *ops*, donde están incluidas tas las operaciones del sistema. Un detalle importante es el nombre que se le pondrá al fichero ya que ese mismo nombre se utilizará para crear el objeto.

   En el fichero *gvc.class.php* hay una parte donde se hace un bucle que se recorre el directorio *ops*, y va generando objetos de cada uno de los ficheros que hay en el mismo pero añadiéndoles delante la palabra *Op*. En nuestro ejemplo, si el fichero se llamara *GestionarFacultades.php* el objeto que se creará en el sistema será *OpGestionarFacultades*. A partir de este momento se podrá utilizar este objeto a través de sus métodos.

Programa A.1: Código para crear el objeto

```
1
2    // Cargar las operaciones
3    foreach(glob(ROOT.'lib/ops/*.php') as $file) {
4        require_once($file);
5        $id = 'Op'.basename($file, '.php');
```

```
6         $this->Ops[$id] = new $id();
7     }
8 ?>
```

Se referncia el objeto en la aplicación, de la forma:

Programa A.2: Código para referenciar el objeto

```
1 'boton1'    =>    array('nombre'    =>'Gestionar Facultades',
2   'title' =>'Gestionar Facultades',
3   'imagen'=>'images/facultad',
4   'Href'  => $gvc->Ops['OpGestionarFacultades']->Accion($_SESSION['id_facultad'])
5         ),
6 ?>
```

3. *Habilitarlo en la tabla ACL.* En este paso ya esta creada la nueva operación en el sistema, pero hay habilitarla para que la pueden utilizar los usuarios del sistema. Como se comento, existe una tabla ACL, que contiene los permisos de los grupos creados. Éste ultimo paso consiste en asignar esa nueva operación a los grupos crados o a los nuevos grupos para que pueden utilizarla.

Los pasos para añadir una acción son exáctamente los mismos pero omitiendo la inclusión de la operación en la base de datos.

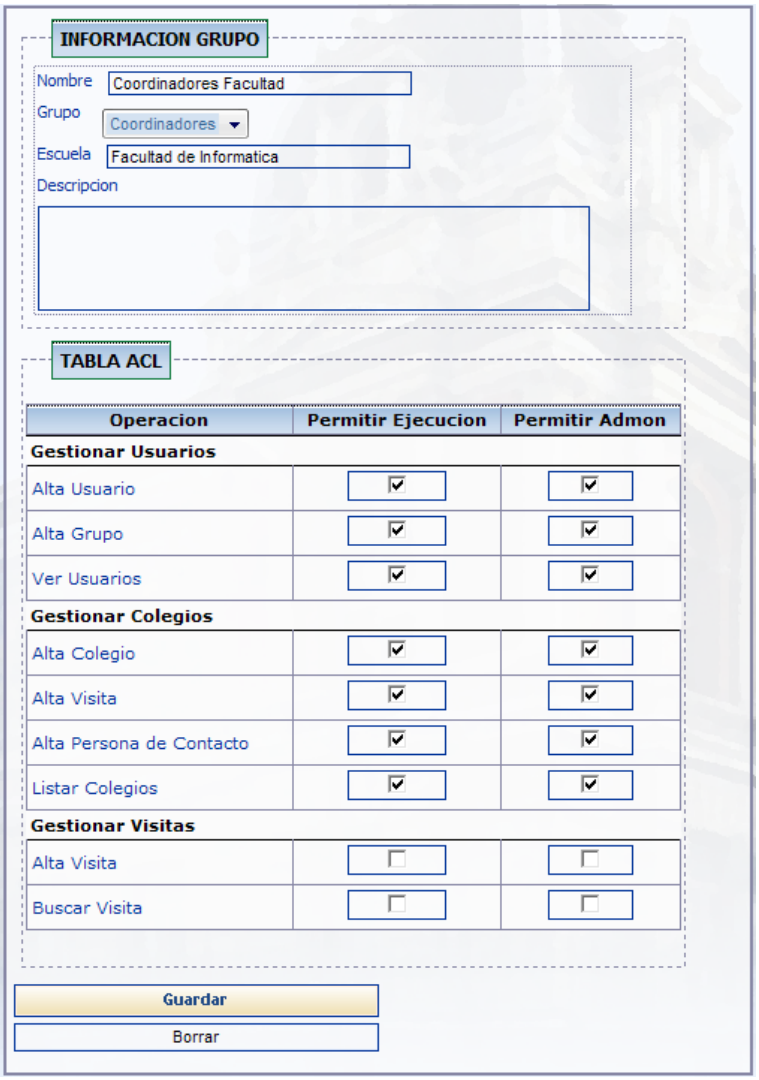


Figura A.3: Habilitarlo en la tabla ACL

Anexo B

Formato de la guía de centros

La Consejería de Educación publica la una guía de centros y servicios educativos de la comunidad de Madrid, la cual es una publicación de divulgación y consulta que trata de facilitar al ciudadano información sobre los servicios educativos de carácter no universitario que se prestan en la Comunidad de Madrid.

La última versión [1] facilita la localización y selección de centros según múltiples criterios, a la vez que ofrece información detallada y precisa sobre centros y servicios educativos de la comunidad. Asimismo, incorpora un sistema de localización y representación geográfica, desarrollado por el Instituto de Estadística de la Consejería de Economía e Innovación Tecnológica, que complementa los recursos para conseguir una mejor y más útil información de los centros y servicios dependientes de la consejería de educación.

La guía se distribuye en formato electrónico en CD. Este CD contiene:

- XML. Contiene archivos en formato XML, con toda la información de los centros:
  - *GENU_DG_DAT.XML*. Contiene información de los centros por las diferentes zonas en la que divide a la comunidad. Estas zonas son: Norte / Sur / Este / Oeste y Centro.
  - *GENU_CE_V_ZONASE.XML*.
  - *GENU_CE_V_TURNOS_ENS.XML*. Contiene la información si los centros imparten enseñanzas presenciales o no.
  - *GENU_CE_V_NIVEL_ENS.XML*. Contiene información a cerca de las enseñanzas que imparte cada centro.

[1] la cual se encuentra en CD y que incorpora una serie de nuevas funcionalidades respecto a la versión 2005

Figura B.1: Página principal de CD de los centros educativos

- *GENU_CE_V_CURSO_ENS.XML*. Contiene información a cerca del nivel de enseñanza de cada centro, es decir, educación infantil, bachillerato, educación primaria, E.S.O, formación profesional de grado medio y superior, idiomas, etc.
- *GENU_CE_V_NIVEL.XML*. Contiene información a cerca del centro, como por ejemplo el numero de alumnos, etc.
- *GENU_CE_V_LOCAL_CRA.XML*. Contiene información de los centros por municipio.
- *GENU_CE_V_JERARQUIA.XML*. Contiene información de las enseñanzas y modalidades que pueden impartir los centros.
- *GENU_CE_CENTROS_DOMICILIO.XML*. Contiene información del domicilio de todos los centros.
- *GENU_CE_V_ENS.XML*. Contiene todas las enseñanzas que se imparten en cada centro.
- *GENU_CE_V_DAT_MUNICIPIO.XML*. Información sobre los municipios de la comunidad por las zonas en las que se divide.
- *GENU_CE_V_CENTROS_DOMICILIO.XML*. Contiene información a cerca de la dirección de algunos centros.
- *GENU_CE_V_CENTROS.XML*. Contiene información de los centros las zonas y su dirección.
- *GENU_CE_V_CENTRO_TRAMO.XML*. Contiene información sobre centros de primaria, secundaria y educación infantil.

- *GENU_CE_V_AMBITO_GEO.XML*. Contiene información de los centros por ámbito geográfico, es decir, Villarejo contendrá todos los centros de Brea, Estremera y Valdaracete.
- *GENU_CE_TRAMO_EDUCATIVO.XML*. Contiene información sobre centros de primaria, secundaria y educación infantil.
- *GENU_DG_V_GENERICO.XML*. Contiene las abreviatura de los centros, por ejemplo, IES; Instituto de enseñanza secundaria.

- *IMAGES*. Contiene las imágenes y documentación de los centros y servicios educativos, en formato PDF.
- *SKIP*. Carpeta de configuración.

Bibliografía

B. W. Boehm. *Software engineering economics*. 1981. [*Citado en pág. 36.*]

Donald Chamberlin. *Using the New DB2: IBM's Object-Related Database System*. 1996. [*Citado en pág. 26.*]

Peter P. Chen. `http://bit.csc.lsu.edu/~chen/chen.html`, 1976. [*Citado en pág. 28.*]

W3C: World Wide Web Consortium. `http://www.w3c.org/`, 2004. [*Citado en pág. 14.*]

Brendan Eich. *JavaScript Bible*. 2001. [*Citado en pág. 14.*]

María Isabel Ferré Grau, Xavier y Sánchez Segura. *Desarrollo Orientado a Objetos con UML*. 2003. [*Citado en pág. 40.*]

H. Gomma. A software design method for real-time systems. *Communications of the ACM, volumen 17*, 1995. [*Citado en pág. 36.*]

James Rumbaugh Grady Booch, Ivar Jacobson. *The unified modelling language*. 1999. [*Citado en pág. 53.*]

C. Larman. *UML y Patrones*. 1999. [*Citado en pág. 38.*]

LaTeX. `http://www.latex-project.org/`, 2004.

PEAR. `http://pear.php.net/`, 2004a. [*Citado en pág. 16.*]

PEAR. `http://pear.php.net/package/DB_Table`, 2004b.

PEAR. `http://pear.php.net/package/HTML_QuickForm`, 2004c. [*Citado en pág. 78.*]

Carles Dorado Perea. `http://www.xtec.es/~cdorado/cdora1/esp/disseny.htm`, 1996. [*Citado en pág. 67.*]

PHP. `http://www.php.net/`, 2004. [*Citado en pág. 15.*]

Fernando Alonso Amo; Loïc Martínez Normand; Francisco Javier Segovia Pérez. *Modelos de desarrollo de programas*. 2002.

W.W Royce. *Managing the development of large software systems: concepts and techniques*. 1970. [*Citado en pág. 35.*]

Abraham Silberschatz. *Fundamentos de Bases de Datos*. 1998. [*Citado en pág. 28.*]

SimManTools. A simple timesheet. `http://simmantools.sourceforge.net/`, 2004.

SMARTY. `http://smarty.php.net/`, 2004. [*Citado en pág. 17.*]

HTML 4.01 Specification. `http://www.w3.org/TR/html4/`, 2004. [*Citado en pág. 13.*]

Jennifer Ullman, Jeffrey y Widom. *Introducción a los Sistemas de Bases de Datos*. 1999. [*Citado en pág. 28.*]

Motor Zend. `http://www.zend.com/zend/whats-new.php`, 1999. [*Citado en pág. 15.*]